Conectografía de la gripe aviar: herramientas para predecir la difusión de la enfermedad
Conectography of the avian influenza: tools to predict the spread of the disease

Trabajo de Fin de Máster
Curso 2018–2019

Autor
Carlos Ballesteros de Andrés

Director
José Ignacio Gómez Pérez
Christian Tomás Tenllado Van Der Reijden

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

Conectografía de la gripe aviar: herramientas para predecir la difusión de la enfermedad
Conectography of the avian influenza: tools to predict the spread of the disease

Trabajo de Fin de Máster en Internet de las Cosas
Departamento de Arquitectura de Computadores y Automática

Autor
Carlos Ballesteros de Andrés

Director
José Ignacio Gómez Pérez
Christian Tomás Tenllado Van Der Reijden

Convocatoria: *Septiembre*
Calificación: *7,5*

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

25 de septiembre de 2019

Dedicatoria

A mis padres, a mi hermana y sobre todo a Irene,
por haber aguantado todos mis gruñidos.

Agradecimientos

A Nacho y Christian por su paciencia infinita en estos seis meses de retrasos y fallos en las previsiones. A Irene Iglesias por todo el material y las buenas ideas facilitadas. A todos los compañeros y sobre todo amigos que con sus ideas y consejos han aportado su granito de arena para llegar aquí.

Resumen

Conectografía de la gripe aviar: herramientas para predecir la difusión de la enfermedad

Existen varios subtipos del virus de la gripe aviar altamente patógeno (HPAI) que circulan de forma global a través de, entre otros medios, los flujos migratorios de las aves acuáticas.

En este proyecto se han estudiado y desarrollado una serie de herramientas que permiten analizar y monitorizar los brotes de HPAI localizados en Europa, así como tratar de modelar los caminos que siguen algunas especies de aves migratorias para generar alertas de forma temprana.

Concretamente se ha diseñado, utilizando diferentes tecnologías, un sistema que monitoriza diariamente los brotes identificados oficialmente en la plataforma *WHAHIS-OIE* y los cruza con el grafo generado a partir de los avistamientos migratorios proporcionados por Sociedad Española de Ornitología, para generar las mencionadas alertas sobre las distintas comarcas ganaderas y a su vez evaluar, en cada caso, el riesgo potencial de contagio.

El sistema se aloja en una aplicación accesible desde una interfaz web y ha dado buenos resultados para los brotes relacionados con las migraciones de las especies estudiadas.

Palabras clave

Python, Flask, MongoDB, Neo4j, Folium, HPAI, Data Science, Web scraping, Avian Influenza

Abstract

Conectography of the avian influenza: tools to predict the spread of the disease

Several subtypes of highly pathogenic avian influenza virus (HPAI) exist and circulate globally through, among other means, migratory flows of waterfowl.

The project aims to study and develop a set of tools that allows analyzing and monitoring the European HPAI outbreaks, as well as trying to shape the paths that some species of migratory birds are following in order to generate early warnings.

Specifically, a system that daily monitors the officially identified outbreaks in the platform *WHAHIS-OIE* has been designed, employing different technologies. It crosses obtained data with the graph generated from the migratory sightings provided by the Spanish Ornithological Society, to generate the above mentioned alerts on the different livestock regions which in its turn assess, in each case, the potential risk of infection.

The system is hosted in an accessible application from a web interface and has given good results for outbreaks related to the migrations of the studied species.

Keywords

Python, Flask, MongoDB, Neo4j, Folium, HPAI, Data Science, Web scraping, Avian Influenza

Índice

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1

Introducción

"Los datos son la nueva ciencia. El Data Science son las respuestas "
— Pat Gelsinger, CEO de VMware

La Grive Aviar (*Avian Influenza* o AI) es una enfermedad de distribución mundial que afecta tanto a aves domésticas como silvestres. Los movimientos de las aves migratorias(especialmente acuáticas) han jugado un papel muy importante en la difusión intercontinental de los subtipos más presentes actualmente en Europa (H5N1 y H5N8), y la transmisión en la interfaz vida silvestre-doméstica es recurrente a pesar de las medidas de bioseguridad y control.

La epidemiología se considera la ciencia básica para la medicina preventiva y se aplica en la ciencia veterinaria para investigar la dinámica y los factores asociados a las enfermedades que afectan a las poblaciones animales. El foco actual de la epidemiología ya no es el tratamiento del animal individual como ocurría hasta los años cincuenta del siglo pasado, ni las mejoras en salud y productividad de los rebaños como se hizo hasta los años noventa. Actualmente el foco es la cadena alimentaria completa y la estandarización de la calidad y seguridad alimentarias dentro de un contexto global que incluye además la salud humana y la de la vida silvestre, conocido como ***One Health***. Se ha hecho evidente que la separación histórica de la vigilancia de enfermedades de animales y personas ignora la conexión inherente de los sistemas biológicos, lo que hace realmente necesario el enfoque de *One health* . Esta visión global de la sanidad requiere estudios científicos más amplios y multidisciplinares como es la unión de la informática y el data science con especialistas sanitarios, como se pretende mostrar en el desarrollo de este sistema de alerta para la detección temprana de ***influenza aviar altamente patógena (IAAP)*** en España.

1.1. Motivación

La epidemia de AI acaecida en Europa en el año 2006, tuvo consecuencias económicas desastrosas para el sector avícola. Tras unos años de calma, desde 2016,val ha habido una nueva oleada de brotes tanto en aves silvestres como domésticas en Europa con más de treinta países afectados, entre ellos España; que se vio afectada en 2017 con 2 brotes en aves silvestres en Palencia y Gerona y un brote en aves domésticas en Gerona, una de las zonas avícolas más importantes del país.

España recibe millones de aves todos los años procedentes de distintos lugares de Europa, como puede verse en la Figura 1.1. Estas migraciones son una amenaza constante de introducción de la AI, lo que supone un riesgo importante para el sector avícola y, en menor medida, para la salud pública de nuestro país.

Hay que destacar que en España, la producción final agraria constituye el 4 % del P.I.B., del cual el 38 % lo representa la Producción Final Ganadera con más de dieciséis mil millones de euros. Destacan el sector avícola que ocupa el segundo puesto en producción de carne en España y en la UE sólo por detrás del Reino Unido.

La importancia de la sanidad animal en España, por su impacto en la producción ganadera, deriva de que esta última es un pilar de la industria agroalimentaria y una de las principales actividades económicas de nuestro país, con un balance comercial exterior positivo y con una tendencia creciente en los últimos años. Por ello resulta preciso mantener las poblaciones ganaderas libres de enfermedades que supongan un riesgo económico para su estabilidad y rentabilidad, evitando así posibles restricciones económicas y perturbaciones de los mercados.

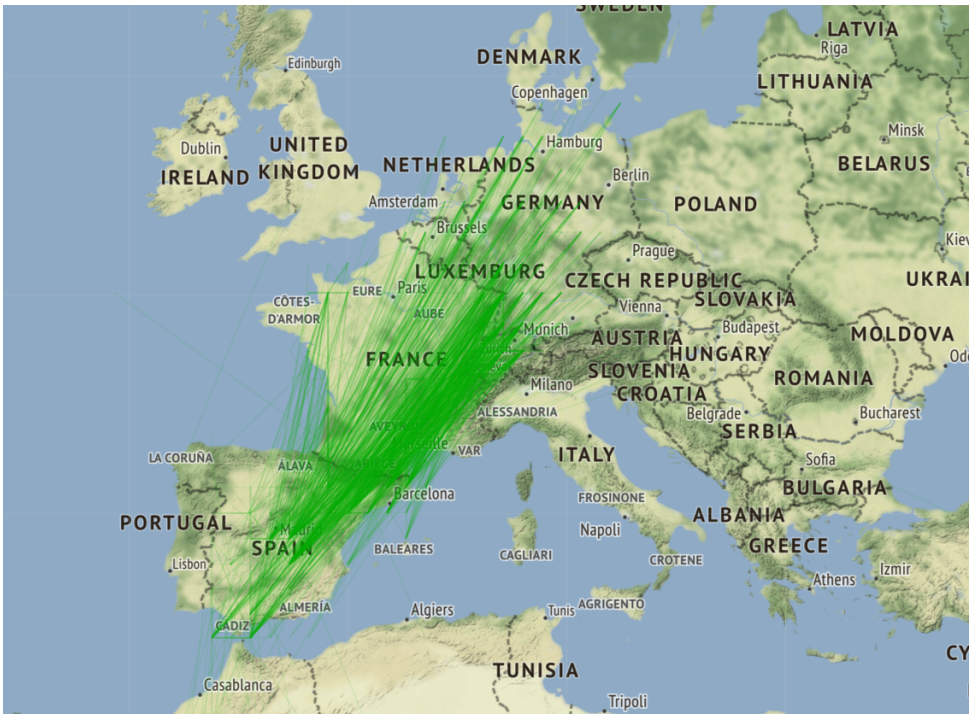

Figura 1.1: Avistamientos migratorios de *Ciconia ciconia* (Cigüeña común) procedentes de Europa con destino a España.

1.2. Objetivos

El principal objetivo del proyecto es desarrollar una herramienta capaz de predecir, con un alto nivel de fiabilidad, alertas de Avian Influenza en España a partir de los brotes documentados en Europa. Para ello se necesita:

- Una fuente de datos actualizada de forma automática, que permita la adquisición y exposición de la información de brotes y migraciones de forma sencilla.

- Un modelo eficiente, tanto en tamaño en disco como en rendimiento, de almacenamiento para la información de brotes y migraciones.
- Una aplicación gráfica para visualizar los avisos. Para permitir una mayor accesibilidad es recomendable que esté hospedada en un servidor web.
- Una interfaz sencilla e intuitiva, dado que la aplicación debe ser utilizada por un abanico muy amplio de usuarios.
- Una interfaz compatible con diferentes tamaños de pantalla, para que pueda usarse tanto en ordenadores, como smartphones y tablets.

1.3. Plan de trabajo

El plan de trabajo para la realización del proyecto ha de seguir las siguientes fases:

- **Análisis del problema**. En esta fase se trata de entender el problema. Se analizarán las distintas fuentes de datos con las que se cuenta para la realización del proyecto así como los requerimientos de los potenciales usuarios del mismo.
- **Diseño de la arquitectura del software**. Llegados a este momento se decidirán los módulos en los que ha de dividirse la aplicación, así como las tecnologías necesarias para el desarrollo de cada uno de ellos.
- **Estudio de las tecnologías a utilizar**. Una vez planificada la arquitectura a desarrollar, se habrán de estudiar y configurar las tecnologías necesarias para hacerlo.
- **Desarrollo del software** En esta etapa se implementarán y unirán todos los módulos resultantes del estudio de la segunda fase.
- **Evaluación de la herramienta** Con el software desarrollado llegaría el momento de testear la usabilidad de la aplicación y sobre todo el sistema de alertas desarrollado. Para ello se utilizarán los brotes de Avian Influenza documentados en España hasta la fecha para comprobar como se habría comportado la herramienta.
- **Redacción de la memoria y documentación del proyecto**. Por último se colectará toda la información adquirida durante el desarrollo para la redacción de la documentación.

1.4. Estructura del documento

El resto de esta memoria se ha dividido en capítulos siguiendo la siguiente estructura:

- En el Capítulo 2 se aborda el estado actual del problema en varios ámbitos, desde la adquisición de los datos en el ámbito concreto de las aves migratorias al diseño de aplicaciones multiplataforma, pasando por librerías y bases de datos para el estudio de grandes cantidades de datos.
- El Capítulo 3 trata sobre los diferentes procesos que se han seguido para adquirir los datos, así como el almacenamiento de los mismos, viendo para esto último las diferentes tecnologías utilizadas.

- Una vez adquiridos los datos, el siguiente punto es explicar el uso que se ha hecho de los mismos, lo cual se detalla en el Capítulo 4.
- En el Capítulo 5 se explican los diferentes pasos seguidos así como las herramientas utilizadas para desarrollar la aplicación en la que se basa el proyecto.
- El Capítulo 6 presenta las conclusiones obtenidas durante y después del desarrollo del trabajo. Además en este capítulo se indican los siguientes pasos o directrices para seguir desarrollando el proyecto.

Capítulo 2

Estado de la Cuestión

"En principio, la investigación necesita más cabezas que medios."
— Severo Ochoa, Médico Español

Antes de comenzar con el desarrollo del proyecto, es necesario hacer un estudio del ámbito del problema así como su estado actual para saber como afrontarlo.

Desde un punto de vista global, como se ha visto en el Capítulo 1 desde el año 2016 se ha producido un incremento en los casos de gripe aviar en Europa. Además al ser España el destino de un gran número de aves migratorias conviene mantener un control riguroso, puesto que la introducción de la enfermedad en el país puede tener un gran impacto socioeconómico (González, 2006).

Ya se han llevado a cabo proyectos que aplican la tecnología del Internet de las cosas (IoT) para detectar brotes de AI de forma temprana. Por ejemplo en Okada et al. (2009) se detalla como instalando *wareables* con sensores de movimiento y temperatura a diferentes aves domésticas se puede detectar la infección bastante antes de que ésta se manifieste de una forma clara.

A continuación se detalla desde el ámbito del proyecto el estado actual de la problemática desde diferentes puntos de vista.

2.1. Obtención de datos de brotes

Existen varios portales, de organismos oficiales, en los que se puede encontrar información acerca de brotes de gripe aviar. Todas ellas tienen en común que no proporcionan una forma sencilla de descargar los datos. Hasta mediados de 2019 estaba disponible el portal **EMPRES-I**[1], diseñado por el sistema de prevención de enfermedades veterinarias del departamento de agricultura de las Naciones Unidas.

La alternativa más actualizada a día de hoy es **WAHIS**[2] iniciativa de la Organización Mundial de Sanidad Animal (OIE). En esta aplicación se registran datos de migraciones a nivel mundial, con el inconveniente de que no permite descargarlos de una forma sencilla.

[1] `http://aims.fao.org/activity/blog/empres-global-animal-disease-information-system`
[2] `https://www.oie.int/wahis_2/public/wahid.php/Wahidhome/Home/index/newlang/es`

Existen diversos proyectos en internet para intentar descargar datos de esta página de los que se puede partir para desarrollar soluciones en este ámbito.

2.2. Estudio de flujos migratorios

El seguimiento de las rutas migratorias y el control de la migración ha sido y es uno de los principales focos de estudio de la ornitología. Los datos de seguimiento pueden ser difíciles de conseguir para ciertas especies de aves migratorias, puesto que aunque la mayoría de las especies son pequeñas, recorren distancias de cientos o miles de kilómetros, con el desafío que supone esto a los investigadores. Existen multitud de estudios que abordan este problema desde diferentes ámbitos, como es el caso de Bridge et al. (2011) o Fiedler (2009) que lo afrontan desde el punto de vista del hardware necesario y las comunicaciones.

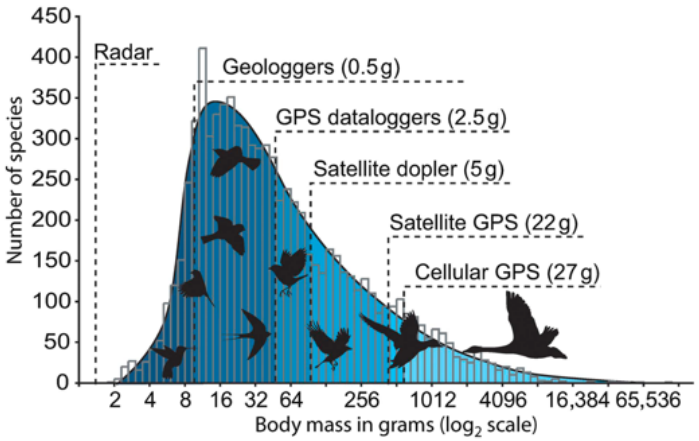

Figura 2.1: Distribución de especies de aves según su masa y métodos de geoposición

Desde finales del siglo XX se han desarrollado tecnologías que facilitan el seguimiento de las rutas migratorias, como dispositivos GPS miniaturizados de menos de un gramo, o el seguimiento vía satélite que propone la **Iniciativa ICARUS**[3]. No obstante cualquier dispositivo que se acople al animal debe de ser lo suficientemente pequeño para que no le afecte pero debe emitir la suficiente energía para ser detectado, lo que lleva a uno de los principales problemas que existe ahora mismo en el ámbito del IoT y es la relación peso/duración de las baterías. Este es un problema que se está solucionando poco a poco con las mejoras tecnológicas que se están desarrollando en este ámbito.

Dentro de las diferentes tecnologías de geolocalización, dependiendo de la masa del ave se pueden elegir unas alternativas u otras. Esto último puede verse en la Figura 2.1 en la que se detalla la distribución de especies de aves según su masa y el mejor método de geoposicionamiento paracada una de ellas. A continuación se explican las cuatro tecnologías mayoritarias en este ámbito:

- **Seguimiento vía satélite:** Son quizás de los más atractivos porque permiten la adquisición en tiempo real de datos en casi cualquier parte del mundo, lo cual permite a los investigadores no sólo seguir a las aves cuando están vivas, si no saber en que lugar mueren.

[3] `https://www.icarus.mpg.de/en`

El principal problema de este método es el alto precio de la infrasestructura necesaria y la energía necesaria para las transmisiones, lo que incide en el peso de los dispositivos, siendo los más pequeños de unos cinco gramos de los cuales casi todo pertenece a la batería.

En este ámbito se encuentra el proyecto ICARUS (International Cooperation for Animal Research Using Space), que mediante una antena en la Estación Espacial Internacional pretende seguir los movimientos migratorios de multitud de especies equipadas con pequeños emisores.

- **Sistemas basados en VHF y redes móviles:** Son sistemas basados en la infraestructura de tierra, sobre todo antenas montadas en torres. Utilizan emisiones de alta frecuencia, por lo que no tienen un largo alcance y no son válidos por ejemplo para seguimientos a nivel internacional. Si se utilizan redes ya existentes, como por ejemplo 4G se puede ahorrar mucho coste en infraestructura a costa de pagar la cuota de transmisión, pero el coste de los dispositivos por contra es más barato que para el caso anterior, encontrándose equipos en el mercado por menos de 100€. Además utilizando redes GSM se simplifica el proceso de geolocalización, pues se puede utilizar como referencia la de la antena desde la que se ha hecho la conexión.

  Se han conseguido implementar dispositivos de seguimiento de menos de 0.5g utilizando esta tecnología, y monitorizar especies bastante pequeñas como pueden ser colibríes. Sin embargo, un sistema combinado GPS+GSM en este momento es dificultoso por el peso necesario para poner en funcionamiento el dispositivo, pudiéndose equipar sólo en individuos grandes como es el caso de la cigüeña de la Figura 2.2. Al igual que en los métodos basados en satélite es necesario mejorar en el desarrollo de las tecnologías de las baterías y la alimentación solar para una implementación masiva de este sistema.

Figura 2.2: Cigüeña equipada con equipo de localización GPS+GSM.

- **Dispositivos de almacenamiento de geoposición:** Una forma de reducir requisitos de energía de los dispositivos de rastreo es eliminar el envío de los datos. De esta forma la información de geoposición es almacenada en el dispositivo y leída del mismo cuando es recuperado por el investigador. En muchas especies la posibilidad de recuperarlos es ínfima, pero hay casos en los que se conoce que las aves vuelven al lugar de origen.

  Dentro de este campo existen dos variedades, los que adquieren los datos por GPS y

los que lo hacen en función de la luz del sol. De los primeros su principal inconveniente es el peso y el precio. Los más utilizados son los que se basan en la luz del sol puesto que son muy ligeros, menos de 0.2g (Naef-Daenzer et al., 2005), y baratos. Su funcionamiento se basa en un reloj de tiempo real y un sensor de luz, de esta forma mediante la duración del día y la noche y la hora a la que se ha producido el mediodía solar se puede aproximar la latitud y longitud a la que está el individuo monitorizado. Un ejemplo de estos sensores puede verse en la Figura 2.3 en la que un *Empavesado pintado* de unos 17 gramos, carga con un geologger solar de unos 0.7 gramos.

- **Radar:** El radar es otro método bastante eficiente para rastrear movimiento de aves, pero es más útil para seguir a bandadas que a individuos concretos. No utilizan ningún tipo de hardware sobre las aves, pero su inconveniente es que la infraestructura para realizar estudios en grandes distancias es muy cara. Gracias a esta técnica se ha podido seguir el movimiento de aves en zonas de difícil acceso, permitiendo además el modelado del terreno (Swatantran et al., 2012).

Figura 2.3: Sistema de geologger montado sobre un ave

Todas estas técnicas conviven con otras más tradicionales, entre las que destaca la **técnica del anillamiento**. El anillamiento de aves se lleva practicando desde principios del siglo XX y consiste en la colocación de una anilla, normalmente de plástico o aluminio en la pata de un ave viva. Esta anilla lleva impresa una serie de información como el lugar y fecha del anillamiento, o la entidad encargada del proceso. De esta forma cuando se reencuentra al mismo individuo en el lugar de origen o en cualquier otro se pueden obtener datos sobre hábitos migratorios, longevidad, estudios de población, etc.

Según la ONG **SEO Birdlife**[4] trabajan en España cerca de 600 anilladores repartidos por todo el territorio. Cabe mencionar que los datos aportados por la Sociedad Española de Ornitología y utilizados en el proyecto se han obtenido con este método.

Se están desarrollando métodos más avanzados para intentar inferir los movimientos migratorios, en el artículo Lin et al. (2019) se cuenta como ya se han podido predecir con bastante precisión datos migratorios utilizando técnicas de inteligencia artificial y en Su et al. (2018) se estudia como modelar os movimientos migratorios a partir del procesado de imágenes y datos climáticos.

[4] `https://www.seo.org/`

Toda la información procedente de estas técnicas es compartida por algunas instituciones en diferentes plataformas como eBird[5], AKN[6] o Movebank[7]

2.3. Bases de datos

Se define como **base de datos** a un conjunto de información perteneciente a un mismo contexto, ordenada de modo sistemático para su posterior recuperación, análisis y/o transmisión. Así mismo llamamos **motor de bases de datos** o **Sistema gestor de bases de datos** al software encargado de gestionar esa información.

Existen diferentes tipos de bases de datos, siendo los dos más importantes:

- **Bases de datos relacionales:** Siguen el modelo relacional, esto quiere decir que permiten establecer interconexiones (relaciones) entre los datos, que se encuentran almacenados de forma estructurada en tablas. Las relaciones se guardan en las tablas y se representan mediante referencias a otras tablas. Algunos ejemplos de este tipo de bases de datos son **MySQL**, **SQL Server** u **Oracle Database**.
- **Bases de datos no relacionales:** También conocidas como NoSQL. No están diseñadas para modelos de datos específicos y tienen esquemas flexibles. Son fáciles de desarrollar, y aportan mucha flexibilidad y escalabilidad. **MongoDB** o **Cassandra**, por citar algunos, son ejemplos de bases de datos no relaciones.

Dentro de las bases de datos no relaciones podemos encontrar muchos subtipos, como las basadas en grafos (**Neo4j**), en clave-valor (**Redis**), o las que contienen documentos y optimizan búsquedas en ellos (**Elasticsearch**)

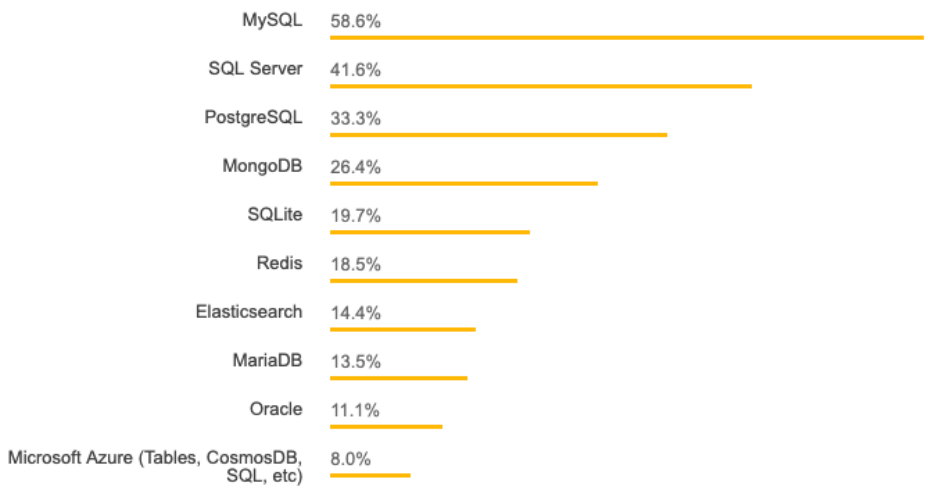

Figura 2.4: Bases de datos más utilizadas en 2019. Fuente: Encuesta de Stackoverflow

Según una encuesta del portal **Stackoverflow**[8], cuyo resultado puede verse en la Figura 2.4 las bases de datos más utilizadas en 2019 han sido, por este orden: **MySQL**, **SQL**

[5] `https://ebird.org/home`
[6] `http://avianknowledge.net/`
[7] `https://www.movebank.org/`
[8] `https://insights.stackoverflow.com/survey/2018/`

Server, **PostgreSQL**, **MongoDB** y **SQLite**

2.4. Lenguajes de programación y librerías en Data Science

Son muchos los lenguajes de programación utilizados actualmente en producción en función de las necesidades y el problema. Según el **Índice TIOBE**[9], elaborado por una empresa de software holandesa que se especializa en la evaluación y seguimiento de la calidad de los programas informáticos, los lenguajes más utilizados son (en este orden) **Java**, **C**, **Python**, **C++** y **C#**. Además, como puede observarse en la Figura 2.5, Python ha experimentado un crecimiento a tener en cuenta los últimos años debido sobre todo a su eficiencia, legibilidad y versatilidad, existiendo una gran colección de bibliotecas para prácticamente cualquier uso.

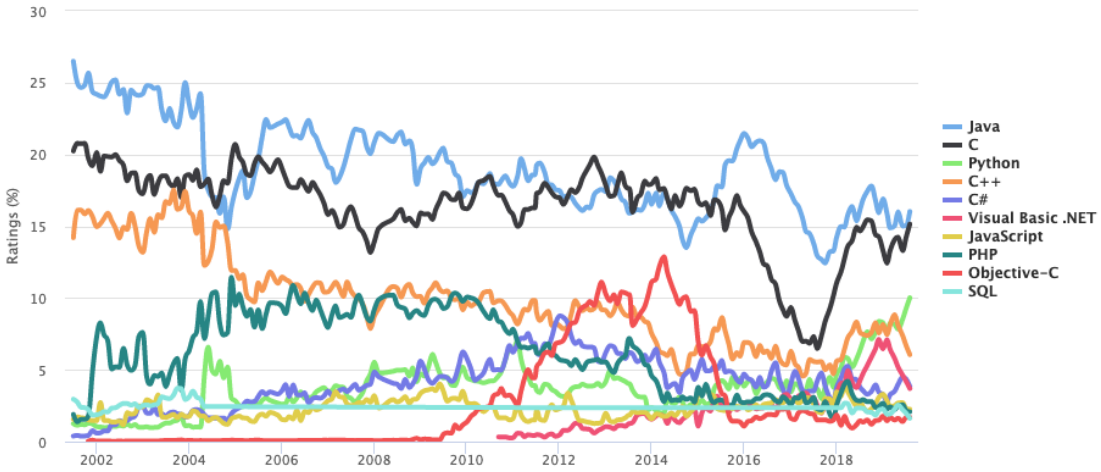

Figura 2.5: Evolución de uso de diferentes lenguajes de programación. Fuente: TIOBE

Muchas de estas bibliotecas están orientadas al tratamiento de datos, existiendo por ejemplo librerías que actúan de conectores con prácticamente todos los motores de bases de datos. Otras más especificas permiten el tratamiento de grandes volúmenes de información de forma sencilla y eficiente como **Pandas** u obtener información de páginas web como **Selenium**.

2.5. Desarrollo de aplicaciones multiplataforma

Se conoce como **aplicaciones multiplataforma** a aquellas que pueden ejecutarse sin problemas en diferentes arquitecturas software y/o hardware.

Hay dos tipos principales de aplicaciones de este tipo, las que se escriben en un lenguaje interpretado (no compilado) como Java, o aquellas que se ejecutan en un servidor y se ofrecen a los clientes para que puedan acceder a ella. En este segundo grupo se incluyen las aplicaciones web. Implementar una solución web para este problema es la mejor opción, puesto que es la opción más sencilla y accesible para el usuario final, que podrá utilizarla tanto desde el ordenador como desde un smartphone o una tablet.

[9] `https://www.tiobe.com/tiobe-index/`

Otra de los factores a tener en cuenta a la hora de construir una aplcación, y más aún cuando se ha dicidido que esta sea accesible desde la web, es el uso de frameworks. Como ya se explicará más adelante un framework básicamente es una herramienta que proporciona una serie de funciones para facilitar la programación. En el ámbito de las aplicaciones web los frameworks más utilizados son **Laravel** y **Symfony** sobre PHP y, lo que más interesa para este proyecto, **Flask** y **Django** sobre Python.

Capítulo 3

Obtención y Almacenamiento de los datos

"Ya no estamos en la era de la información. Estamos en la era de la gestión de la información."
— Chris Hardwick, actor.

Se procede a explicar en este capítulo todo lo relativo a la obtención y almacenamiento de los datos. Para ello se han de estudiar los motores de bases de datos utilizados, así como el modelado y tratamiento que se ha hecho de la información para un eficiente uso de la misma

3.1. Obtención de los datos de brotes. Web Scraping

Hay numerosas formas de obtener la información de los sitios web, lo más habitual es que las instituciones públicas proporcionen sistemas basados en API REST o SOAP para obtenerla, pero no siempre es así. Como se menciona en el Capítulo 2, existen diversos portales con información pública acerca de brotes de gripe aviar. Para este proyecto se ha decidido utilizar la información contenida en el portal WAHIS-OIE[1], al ser la más completa y actualizada de la que se dispone, pero esta web no facilita ninguna opción para descargarlos, por lo que se ha tenido que recurrir al **Web Scraping**.

El web scraping es una técnica que consiste en emular de forma automática el comportamiento de una persona a través de una web para extraer la información que hay en ella, lo cual requiere un estudio previo del código y estructura de la página para optimizar el comportamiento del script o programa que la ejecuta.

Existen multitud de aplicaciones y librerías para la obtención automática de datos. En Python una de las más utilizadas es **Selenium**[2], pero para este proyecto y dada que la estructura de la información en el portal WAHIS-OIE es bastante clara, se ha optado por realizar el scraping a través de expresiones regulares. Esta técnica consiste en realizar una petición *post* para traernos el código de la página y buscar patrones de texto y coincidencias en él.

Se va a explicar con un ejemplo como se ha realizado el proceso de scrapeo en la

[1] `http://www.oie.int/wahis_2/public/wahid.php/Diseaseinformation/Immsummary`
[2] `https://selenium-python.readthedocs.io/`

aplicación. En el Código 1 se puede ver un ejemplo de tabla extraído de una de los informes de WAHIS-OIE.

En el Código 2 se referencia una parte del script que se ha utilizado para extraer la información. En él se hace una petición post, a la que se incluyen una serie de parámetros, mediante la librería *request* para obtener el contenido de la página en texto plano HTML.

Por otro lado se compila la expresión regular en la se están buscando tres posibles cadenas:

- Una que venga a continuación de una celda con el texto 'start of the event'y con un formato de fecha (dos dígitos/ dos dígitos/ cuatro dígitos).
- Una cadena de caracteres alfabéticos (a-Z) de tamaño ilimitado. A continuación de una celda de tabla con el texto 'Outbreak Status'
- Por último, otra celda con formado de fecha.

```
<table>
    <tbody>
        <tr>
            <td>Date of start of the event</td>
            <td>14/01/2018</td>
        </tr>
        <tr>
            <td>Outbreak Status</td>
            <td>Resolved</td>
        </tr>
        <tr>
            <td>Date of resolution of the outbreak</td>
            <td>10/06/2019</td>
        </tr>
</table>
```

Código 1: Ejemplo de tabla con información del portal WAHIS

El resultado de ejecutar esta función si la página contuviera el ejemplo del código 1 sería una lista con los elementos *'14/01/2018'* , *'Resolved'* y *'10/06/2019'*. En el Apéndice A se puede ver un ejemplo del script utilizado para scrapear todos los datos de AI ocurridos en los países de Europa y Asia desde el año dos mil cinco.

Además, como se explica en el Capítulo 4, se ha utilizado para alojar la información un servidor con el sistema operativo Linux. Los sistemas Linux tienen por defecto un servicio o *demonio* llamado **crontab** que ejecuta procesos en segundo plano de forma programada.

Existe otro script similar al mencionado anteriormente, ejecutado diariamente mediante el crontab del sistema operativo del servidor, que extrae del portal las referencias de los nuevos brotes que se van documentando.

```
def extract_data():
url = 'http://www.oie.int/wahis_2/.../Immsummary/outbreakreport'
r = requests.post(url, data={'reportid':id, 'summary_country':cty})
p = re.compile('start of the event</td>[^>]+>(\d{1,2}/\d{1,2}/\d{4}).*?'
           'Outbreak Status</td>[^>]+>(\w+).*?'
           'resolution of the outbreak</td>[^>]+>(\d{1,2}/\d{1,2}/\d{4})?.*?'
           ,re.DOTALL & re.MULTILINE * re.IGNORECASE)
m = p.findall(r.content.decode('latin1'))
return m
```

Código 2: Ejemplo scrapeo mediante expresiones regulares

3.2. Obtención de los datos de migraciones

En cuanto a los datos de migraciones, como se ha visto en el Capítulo 2 se han estudiado y consultado multitud de páginas de asociaciones y sociedades de ornitología, pero no se ha encontrado una forma factible de automatizar el proceso de carga.

La sociedad Española de Ornitología ha facilitado un fichero CSV con datos de más de veinte mil avistamientos de migraciones en Europa, en las que el destino es España, de los últimos treinta años. Todos ellos obtenidos mediante el método de anillamiento (ver Capítulo 2).

Cada fila de este fichero contiene la información de un avistamiento con los datos (fecha y localización) del anillamiento y del lugar donde se ha vuelto a avistar el ave. Para el modelado de las migraciones se ha supuesto que si un individuo es anillado en un punto A y ha sido avistado en el punto B, existe una ruta migratoria entre los puntos A y B.

Para insertar este fichero en la base de datos, se ha requerido de un preprocesamiento para homogeneizar el estilo de las celdas, eliminar filas no relevantes y dar los formatos correctos a las coordenadas y fechas. Este preprocesado se realizó cargando el fichero a un dataframe mediante la librería de análisis de datos **Pandas**[3], la cual ofrece multitud de funciones para el tratamiento de grandes cantidades de datos en el formato fila-columna, como operaciones masivas en filas o mapeos a nivel de columna. Puede verse todo este proceso en el Apéndice B

3.3. Geohash

Tanto la los datos de brotes como los datos de avistamientos migratorios tienen información de localización a diferentes niveles (localidad, provincia, coordenadas...). Al ser necesario realizar operaciones de búsqueda sobre áreas más o menos amplias, se han estudiado diferentes opciones como trabajar operando con radios sobre las coordenadas o a nivel de provincia o región, pero se ha optado por utilizar la técnica del **geohashing**[4].

Geohash es un sistema de geocodificación de dominio público inventado por Gustavo Niemeyer en el año 2008, con el fin de obtener un identificador único para coordenadas geográficas y poder utilizarlo en direcciones de correo, URLs y otras aplicaciones informá-

[3] https://pandas.pydata.org/
[4] http://geohash.org/site/tips.html

ticas. De esta forma a partir de una tupla de coordenadas se obtiene una cadena de letras y dígitos.

Esta técnica, explicada a alto nivel, consiste en dividir el mapamundi en en una cuadrícula de treinta y dos celdas, formada por cuatro filas y ocho columnas. Cada una de estas celdas puede ser a su vez dividida de nuevo en otras treinta y dos celdas, y así sucesivamente.

La primera cuadrícula de todas tiene una precisión apróximada de unos 5000kms x 5000kms y si se va dividiendo cada cuadrícula se obtienen celdas de 1250kms x 625kms, 156kms x 156kms, etc. De una forma similar a la representada en la Figura 3.1.

Figura 3.1: Esquema de cuadrícula del sistema Geohash

Un ejemplo práctico, la Facultad de Informática de la Universidad Complutense de Madrid está localizada en las coordenadas 40.4528582N, 3.7361562O. Estas coordenadas están representadas por la cadena *ezjq4upqvr21*, que tiene doce caracteres lo que quiere decir que se ha dividido la cuadrícula doce veces. En la Tabla 3.1 se muestra la precisión que se puede obtener en función del número de caracteres de la cadena.

Con esta explicación a alto nivel del sistema, es sencillo entender por qué se ha elegido este método de geoposicionamiento para el proyecto. Al tener un orden jerárquico de cuadrículas o regiones se simplifica el proceso de buscar puntos cercanos puesto que basta con que coincida el principio de la cadena que representa la localización de dichos puntos, siendo más cercanos cuantos más caracteres de la cadena coincidan.

Precisión	Dimensiones
12	3.7cm x 1.8cm
11	14.9cm x 14.9cm
10	1.19m x 0.60m
9	4.78m x 4.78m
8	38.2m x 19.1m
7	152.8m x 152.8m
6	1.2km x 0.61km
5	4.9km x 4.9km
4	39km x 19.5km
3	156km x 156km
2	1251km x 625km
1	5004km x 5004km

Tabla 3.1: Precisión de geohash en función de la longitud de la cadena

Este método es ampliamente utilizado en operaciones de análisis de datos, puesto que al ser representada la geoposición como un único *String* de longitud fija, se simplifican en gran medida las búsquedas en bases de datos al poder indexar un solo campo.

Para la generación, de los hashes de los brotes y migraciones se ha utilizado la biblioteca de Python Geohash[5] que ofrece funciones para codificar y decodificar coordenadas con este sistema. Hay que mencionar que esta biblioteca tiene un pequeño bug y requiere algún pequeño cambio en uno de sus códigos, como se explica en el Apéndice C, por lo que en un futuro no sería descabellado pensar en utilizar otra de las librerías publicadas en PyPI que ofrecen estas funcionalidades.

3.4. Motores de bases de datos

A la hora de almacenar la información, se evaluaron varios motores de bases de datos, tanto relacionales como no relacionales. Del primer grupo se valoró utilizar el sistema PostgreSQL[6] debido sobre todo a su gran rendimiento y a la gran variedad de tipo de datos que ofrece, siendo los más relevantes para este proyecto todos los relativos a geoposicionamiento, coordenadas y rutas entre puntos.

No obstante, dada la gran incertidumbre que existía al comienzo del desarrollo con el formato de los datos, se optó por utilizar MongoDB para almacenar la información de brotes y migraciones, ya que ofrece una gran flexibilidad al almacenarse los datos como diccionarios. Más adelante, cuando se vió que tener los datos de migraciones en MongoDB no era excesivamente eficiente se estudiaron otros formatos más acordes a la problemática, eligiendo la base de datos orientada a grafos Neo4j.

3.4.1. MongoDB

MongoDB es una base de datos orientada a documentos. Lo cual quiere decir que los datos se guardan en documentos en vez de en registros. Estos documentos se guardan en

[5] `https://pypi.org/project/Geohash/`
[6] `https://www.postgresql.org/`

formato BSON, que es una representación binaria de JSON.

Los datos se organizan en colecciones, que es un concepto similar a una tabla en una base de datos relacional, pero a diferencia de estas en MongoDB no es necesario seguir ningún esquema. Los documentos de una misma colección pueden tener estructuras diferentes con más o menos campos dependiendo del elemento.

Las consultas en MongoDB se hacen pasando objetos JSON como parámetro. Por ejemplo, una búsqueda que traiga todos los brotes de España sería mediante la orden:

```
outbreaks.find({"country" : "ESP"})
```

Una query algo más compleja, que además filtrara entre dos fechas seguiría el siguiente esquema:

```
outbreaks.find({"country": "ESP",
                "start":{"$gte": "2005-11-15T00:00:00Z",
                         "$lte":"2005-12-27T00:00:00Z" }})
```

En el proyecto se utilizan dos colecciones en MongoDB, que se detallarán más adelante: **outbreaks** para almacenar los casos de Avian Influenza y **migrations** para guardar la información relativa a los datos de avistamientos.

3.4.2. Neo4j

Como se ha comentado anteriormente, la primera idea fue cargar los datos de migraciones en MongoDB, pero aparecieron diferentes problemas:

- No era eficiente en las búsquedas.
- Imposibilidad para realizar algunas operaciones básicas para la aplicación, como la búsqueda de caminos o ciclos.
- Dificultad para visualizar los datos de una forma sencilla.

Es por estos motivos que se realizó un estudio de diferentes sistemas gestores de bases de datos y el elegido fue **neo4j**. Neo4j es un motor de persistencia (los datos se guardan en disco) escrito en Java que almacena los datos estructurados en grafos en lugar de en tablas.

Un grafo se define como una representación gráfica formada por vértices (para este proyecto regiones) y aristas o relaciones (en este caso avistamientos de una determinada especie). Dentro de los diferentes tipos de grafos, se ha utilizado un **grafo con propiedad**, lo cual quiere decir:

- Tiene peso, las relaciones entre nodos tienen una valoración numérica.
- Usa etiquetas, es decir se pueden definir diferentes tipos de vértices y relaciones entre ellos, como las migraciones de diferentes especies.

- Usa propiedades, información libre que se puede añadir tanto a vértices como aristas. Por ejemplo para las regiones se añaden propiedades de localización como la región o país a la que pertenece.

Neo4j es compatible con multitud de lenguajes de programación, además ofrece dos interfaces al usuario para operar directamente sobre las bases de datos. Una es mediante terminal, y otra a través de una aplicación web. Quizás la parte más interesante de la aplicación web sea la visualización de los datos. Las querys se realizan en un lenguaje propio llamado **Cypher** y los resultados de la ejecución de las mismas pueden verse tanto de forma gráfica como en tabla.

3.5. Modelado de la información

Se han detallado hasta este punto tanto los procesos para la obtención de los datos como los sistemas gestores de bates de datos elegidos para almacenarlos, a continuación se detalla la organización de los mismos.

3.5.1. Brotes

Se ha mencionado anteriormente que la información acerca de los casos de gripe aviar se encuentra almacenada en MongoDB. En este caso se haya en una colección llamada **outbreaks**. Outbreaks almacena los datos de los aproximadamente 6500 brotes de AI documentados en Europa y Asia. La información de Asia se obtiene para una posible ampliación de la dimensión del proyecto en un futuro.

Cada elemento de esta colección, con algunas diferencias puesto que no todos tienen los mismos campos, siguen una estructura similar a la del diccionario JSON del Código 3. La información relevante que contiene cada elemento de esta colección es:

- Información de localización tanto a nivel político (localidad, región, país...) como geográfico (coordenadas y geohash).
- Información temporal del brote, fechas de inicio y fin del mismo.
- Información varia sobre los individuos afectados, la cual varía mucho en función de cada elemento de la colección. En este ámbito podemos encontrar por ejemplo el número de animales muertos, la cantiad de inviduos afectados o textos ampliando los datos sobre el brote.

3.5.2. Migraciones

Los datos de las migraciones se encuentran divididos en dos motores de bases de datos. Por un lado existe una colección en MongoDB llamada **migrations** en la que se han ingestado de forma literal los datos del CSV proporcionado por la Sociedad Española de Ornitología y por otro tenemos una representación de los flujos migratorios en forma de grafo almacenada en Neo4j.

La colección migrations cuenta con cerca de 17000 elementos de diferentes especies, de los cuales se han estudiado para el diseño de las alertas los 9700 correspondientes a

```
{
        "_id" : ObjectId("5cd5125005153230453b8cb7"),
        "oieid" : "2633",
        "diseade_id" : "15",
        "country" : "ESP",
        "start" : ISODate("2006-06-30T00:00:00Z"),
        "status" : "Resolved",
        "end" : ISODate("2006-08-07T00:00:00Z"),
        "city" : "Pais",
        "district" : "ÁLAVA",
        "subdistrict" : "VITORIA",
        "epiunit" : "Not",
        "location" : "Embalse",
        "lat" : "42.8619",
        "long" : "-2.6503",
        "affected_population" : "A great crested grebe (Podiceps cristatus)",
        "species" : "Wild species",
        "at_risk" : "1",
        "cases" : "1",
        "deaths" : "1",
        "preventive_killed" : "0",
        "geohash" : "ezw5gdy0et30"
}
```


Código 3: Elemento de la colección **outbreaks**

las migraciones de cigüeña y los 50 de gansos y es la fuente de datos utilizada para la generación de los mapas de histórico. Todos los objetos de esta colección siguen la misma estructura, que se puede observar en el diccionario del Código 4.

La información de los elementos de migrations se encuentra dividida en tres partes:

- Información acerca del animal anillado como su especie, sexo o edad, así como el identificador de la anilla que se le ha puesto.
- Información acerca del lugar de anillamiento, así como la fecha del proceso.
- Información acerca del lugar de recuperación del ave, así como la fecha del proceso.

Con todos los datos mencionados en la colección anterior se ha tratado de modelar el flujo migratorio de las dos especies estudiadas utilizando un grafo valorado alojado en Neo4j que sigue la siguiente estructura:

- Se ha considerado como un nodo a una región de un tamaño de cuatro dígitos en geohash (40km x 20km). Por lo tanto todos los lugares de anillamiento tienen asociados un nodo, así como todos los lugares de recuperación. Cada nodo está referenciado por las cuatro cifras que indican su geoposicionamiento, y tiene información adicional como la provincia y el país al que pertenece.
- Cada arista o relación entre nodos representa a un avistamiento de un ave de una determinada especie que ha sido anillada en un nodo y recuperada en otro. Así

```
{
        "_id" : ObjectId("5cdc220205153261cb3d3593"),
        "Especie" : 1610,
        "FechaAnillamiento" : ISODate("1992-10-23T00:00:00Z"),
        "Localidad" : "LAGUNA DE GALLOCANTA",
        "Municipio" : "GALLOCANTA",
        "Provincia" : "ZARAGOZA",
        "AnillaR" : "G..17682",
        "SexoR" : "U",
        "EdadR" : "0",
        "FechaRecuperacion" : ISODate("1994-07-11T00:00:00Z"),
        "LocalidadR" : "BAGO",
        "MunicipioR" : "ASSENS",
        "ProvinciaR" : "SYDDANMARK",
        "Lat" : "+40.59",
        "Long" : "-1.31",
        "LatR" : "+55.18",
        "LongR" : "+9.48",
        "geohash" : "ezpnfjnjfu3j",
        "geohashR" : "u1y9ztpnu183"
}
```


Código 4: Elemento de la colección **migrations**

tenemos dos tipos de aristas, la nombrada como "MIGRA1610" para las migraciones de gansos y "MIGRA1340" para las migraciones de cigüeñas.

Además al ser un grafo valorado cada arista tiene un determinado valor o peso, en este caso el valor de cada arista es el número de avistamientos que existen para una determinada especie entre dos regiones o nodos.

Para generar este proceso a partir de la colección migrations se ha diseñado un script en Python que obtiene el código Cypher necesario para crear el grafo y lo ejecuta en el servidor.

Una representación visual del grafo que representa a las migraciones de ganso es la que puede verse en la Figura 3.2.

Si se quisiese obtener información más concreta, como la localidad, municipio y provincia de todos los nodos a los que exista una ruta migratoria desde el nodo etiquetado como "u0vf" además del número de avistamientos se podría ejecutar la siguiente query, muy similar a la utilizada para generar las alertas, que devolvería una tabla como la de la Figura 3.3:

```
MATCH p=(a:Region{location:"u0vf"})-[r]->(b) RETURN r.valor,
b.location, b.localidad, b.municipio, b.provincia, type(r)
```

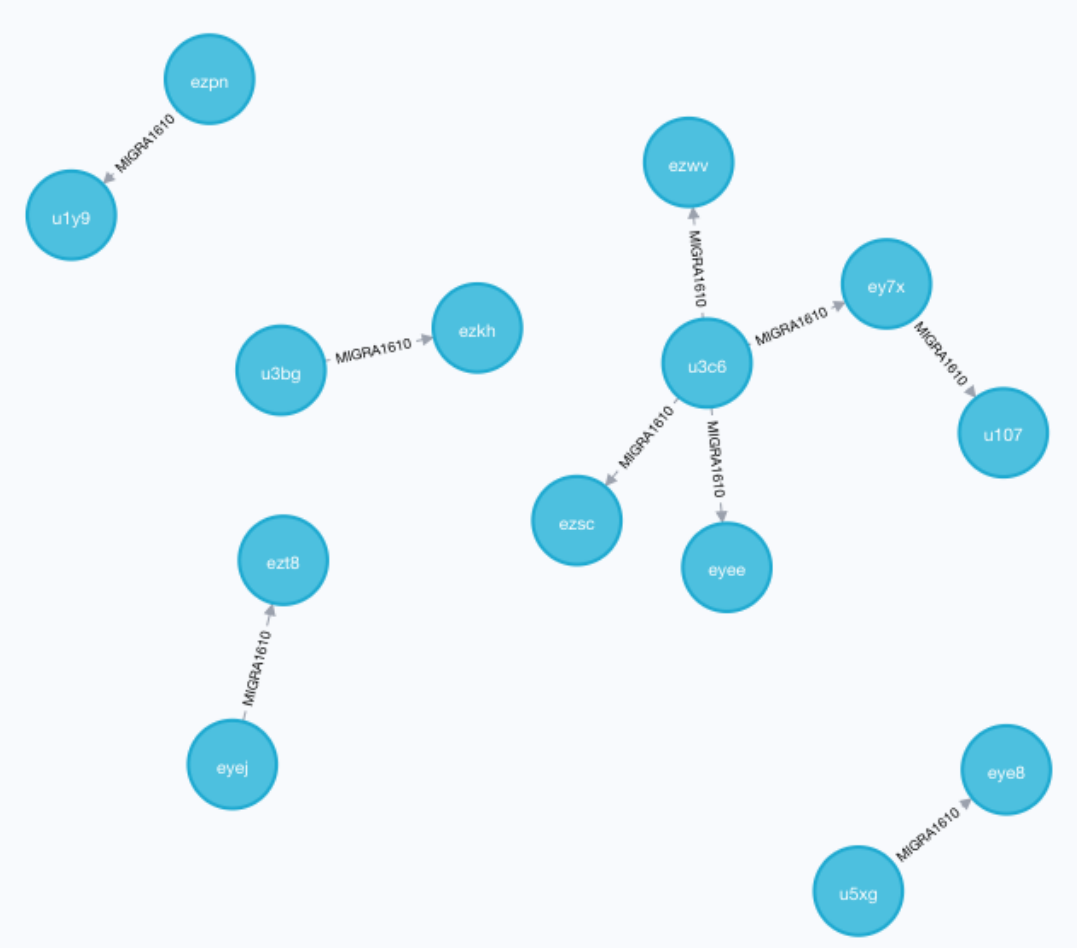

Figura 3.2: Modelado de migraciones en Neo4j

r.valor	b.location	b.localidad	b.municipio	b.provincia	type(r)
2	"sp3z"	"CELRA"	"CELRA"	"GIRONA"	"MIGRA1340"
1	"eysp"	"LOS PALACIOS Y VILLAFRANCA"	"LOS PALACIOS Y VILLAFRANCA"	"SEVILLA"	"MIGRA1340"
4	"ezjt"	"ALCALÁ DE HENARES"	"ALCALÁ DE HENARES"	"MADRID"	"MIGRA1340"
11	"sp6r"	"ABOCADOR DE PEDRET I MARZA"	"PEDRET I MARZA"	"GIRONA"	"MIGRA1340"
1	"eykj"	"V.R.S.U. DE LOS BARRIOS"	"LOS BARRIOS"	"CADIZ"	"MIGRA1340"
1	"ezpn"	"CALAMOCHA"	"CALAMOCHA"	"TERUEL"	"MIGRA1340"
2	"sp9d"	"LLACUNA DE CASA NOSTRA"	"BANYOLES"	"GIRONA"	"MIGRA1340"
1	"sp3w"	"MANLLEU"	"MANLLEU"	"BARCELONA"	"MIGRA1340"
38	"eyvy"	"FABRICA DE URALITA-POLIGONO INDUSTRIAL"	"ALCAZAR DE SAN JUAN"	"CIUDAD REAL"	"MIGRA1340"
3	"eyve"	"ALMAGRO"	"ALMAGRO"	"CIUDAD REAL"	"MIGRA1340"
6	"ey7p"	"V.R.S.U. DE MIRAMUNDO"	"MEDINA SIDONIA"	"CADIZ"	"MIGRA1340"
2	"eyzz"	"PLANTA DE COMPOSTAJE DE FERVASA"	"CUART DE POBLET"	"VALENCIA"	"MIGRA1340"
1	"ey7n"	"CHICLANA DE LA FRONTERA"	"CHICLANA DE LA FRONTERA"	"CADIZ"	"MIGRA1340"
1	"sp2t"	"RIERA ANIMA BLANCA"	"BOTARELL"	"TARRAGONA"	"MIGRA1340"
3	"ezjs"	"PINTO"	"PINTO"	"MADRID"	"MIGRA1340"
2	"sp3d"	"BARCELONA"	"BARCELONA"	"BARCELONA"	"MIGRA1340"
21	"sp25"	"ABOCADOR DE MONTOLIU"	"MONTOLIU DE LLEIDA"	"LLEIDA"	"MIGRA1340"
2	"sp94"	"BERGA"	"BERGA"	"BARCELONA"	"MIGRA1340"
4	"ezre"	"RIO EBRO"	"ALFAJARIN"	"ZARAGOZA"	"MIGRA1340"
1	"sp27"	"IVARS D-URGELL"	"IVARS D-URGELL"	"LLEIDA"	"MIGRA1340"
1	"sp3x"	"OLOT"	"OLOT"	"GIRONA"	"MIGRA1340"
7	"sp3s"	"CALLDETENES"	"CALLDETENES"	"BARCELONA"	"MIGRA1340"

Figura 3.3: Ejemplo de visualización en tabla en Neo4j

Capítulo 4

Diseño y Arquitectura de la aplicación

"A fuerza de construir bien, se llega a buen arquitecto."
— Aristóteles

Llegados a este punto es buen momento para hacer un análisis de los diferentes componentes que han de formar la herramienta:

- Un script que scrapea la página de WAHIS-OIE todas las noches actualizando la información de brotes de Avian Influenza de Europa y Asia en la base de datos, explicado en el Capítulo 3.
- Dos bases de datos, en **MongoDB** y **Neo4j** que almacenan datos de casos de gripe aviar y migraciones respectivamente, también explicados en el Capítulo 3.
- Un módulo **Python** que genera alertas en función de los brotes anteriores y los avistamientos migratorios, que se explicará en el Capítulo 5.
- Otros dos módulos en Python que, usando la librería **Folium** renderizan los mapas de alertas e histórico, de los cuales se hablará más adelante en este capítulo.

Se hace patente que es necesario desarrollar un a aplicación que "encaje" todas estas piezas, lo que ha llevado a una solución cuyo esquema puede verse en la Figura 4.1

Dicha aplicación se encuentra alojada en un servidor virtualizado proporcionado por la Universidad sobre un sistema operativo Linux y unas características de hardware bastante limitadas pero suficientes para ejecutar la herramienta.

4.1. Flask

Uno de los requerimientos previos del proyecto es que la aplicación desarrollara se pudiese ejecutar independientemente de la plataforma, por lo que se pensó desde un momento hacerla accesible vía web. Además al haberse desarrollado la mayor parte de la lógica en Python se estudiaron los diferentes frameworks web que operan sobre dicho lenguaje.

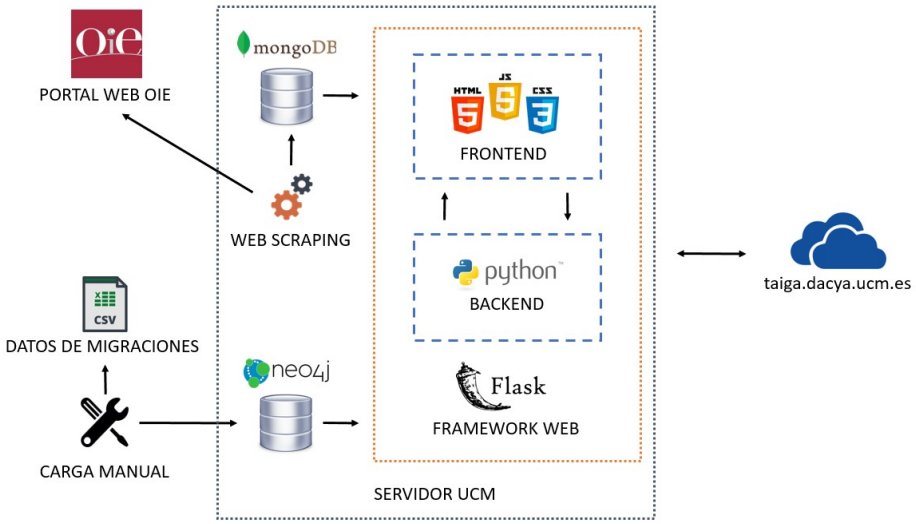

Figura 4.1: Esquema general de la solución propuesta

Los dos frameworks más utilizados sobre Python son **Django**[1] y **Flask**[2], siendo quizás el primero el más potente y el que más funcionalidades ofrece. No obstante Flask ofrece todas las funcionalidades que se necesitaban para el proyecto, además de ser más fácil de utilizar y ligero, por lo que resultó elegido.

Antes de explicar qué es Flask resulta conveniente explicar qué es un framework. Un framework es una herramienta que da una serie de utilidades y funciones así como un esquema de trabajo que facilitan y abstraen al programador en la construcción de aplicaciones.

Sabiendo esto, **Flask** es un framework muy ligero, escrito en Python que permite desarrollar en pocas líneas de código aplicaciones web dinámicas siguiendo el patrón Modelo-Vista-Controlador.

La estructura de una aplicación en Flask es más o menos siempre la misma. En la Figura 4.2 se puede ver el esquema de ficheros de la aplicación desarrollada, con los siguientes elementos a destacar:

- En el directorio **raíz** se encuentran ficheros Python útiles para la aplicación, por ejemplo en nuestro caso tenemos los módulos necesarios para la gestión de las conexiones con la base de datos, la función principal que arranca el sistema (app.py), o los scripts de creación de mapas y alertas.

- En **public** están los códigos en Python que contienen aquellos elementos accesibles desde cualquier parte del sitio web, en este caso los formularios y la configuración de las vistas. Existe una vista diferente para cada url de la página, cuya configuración está en views.py

- **Static** contiene todo los elementos estáticos de la aplicación, véase imágenes, ficheros con contenido JavaScript y las hojas de estilo en CSS3.

[1] `https://www.djangoproject.com/`
[2] `https://flask.palletsprojects.com/en/1.1.x/`

Figura 4.2: Estructura de ficheros de la aplicación

- La carpeta **templates** contiene los ficheros HTML5 templatizados con Jinja2 (que se explicará más adelante). Existe una plantilla para cada página, reutilizándose componentes comunes como el menu lateral, el footer o el layout general.

Flask es el encargado de relacionar los diferentes módulos diseñados en Python con la capa de frontend y publicar el conjunto como aplicación web accesible desde la nube.

4.2. Conectores de las Bases de datos

Se conocen como conectores a las librerías que actúan como capa intermedia entre un lenguaje de programación y los diferentes motores de bases de datos.

Para las operaciones en MongoDB se ha elegido la biblioteca desarrollada y recomendada de forma oficial **pyMongo**[3], que permite con una sintaxis similar a la de la consola de MongoDB realizar todas las operaciones permitidas por el sistema.

Este conector es el que se usa para leer la colecciónes outbreaks, utilizada tanto en el módulo de generación de alertas como en la vista del histórico de brotes, y migrations, de donde se extraen los datos para el histórico de migraciones.

[3] `https://api.mongodb.com/python/current/`

Para operar con Neo4j se ha optado por utilizar **Py2neo**[4], también reocomendado en el sitio oficial de la base de datos. Py2neo añade una capa de abstracción a la base de datos, conocida como **OGM** (Object-Graph Mapping), que añade una serie de facilidades al mapear los nodos del grafo y sus relaciones como objetos de una determinada clase cargándolos en memoria, lo cual simplifica considerablemente el desarrollo a alto nivel. No obstante en esta aplicación al no realizarse demasiados cálculos en tiempo real con los nodos, se ha optado por realizar directamente operaciones en lenguaje Cypher tanto para la generación de alertas como para la creación del grafo de migraciones en el script encargado, siendo las querys cadenas de texto que se crean y modifican en tiempo de ejecución.

4.3. Frontend

Para la implementación del frontend se ha diseñado una página web con un diseño adaptativo que se adapta tanto a una pantalla de ordenador como a las de móviles y tablets. La aplicación cuenta con una vista principal, que puede verse en la Figura 4.3, y vistas para los mapas de alertas e histórico. Además se han creado vistas estáticas para la página *about* y las redirecciones de los errores 404, 401 y 500 que se producen por ejemplo cuando se intenta acceder a una url que no está configurada.

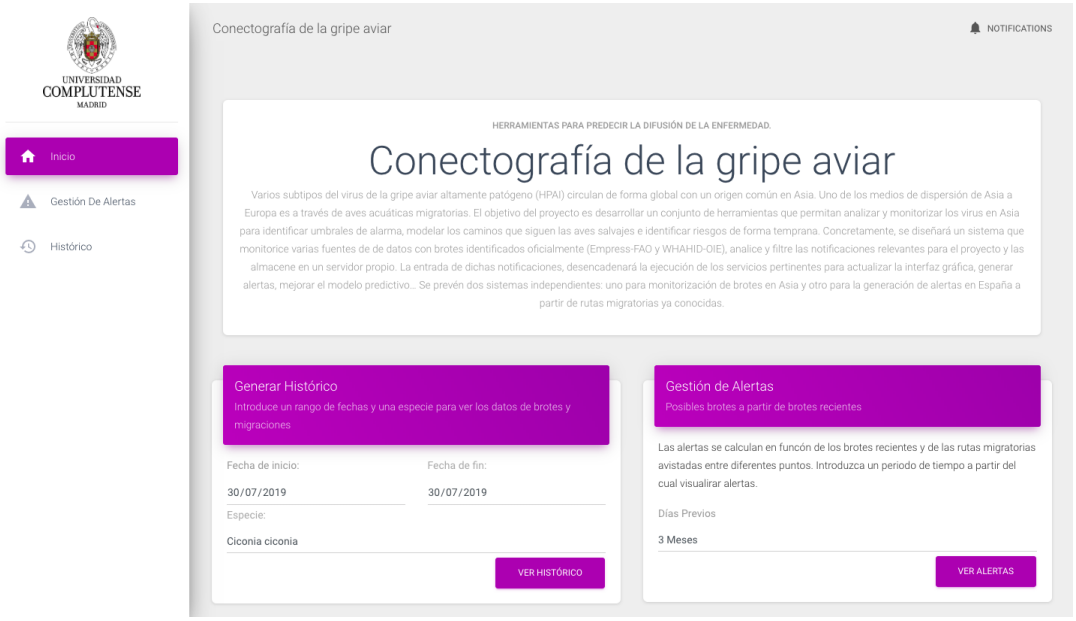

Figura 4.3: Página principal de la aplicación

Se ha utilizado la guía de diseño **Material Design** para los colores, fuentes, iconos y estructura de la aplicación con el fin de mejorar la afabilidad de la herramienta con el usuario.

En esta sección además se detalla el proceso y librerías utilizadas para la generación de mapas

[4] `https://py2neo.org/v3/index.html`

4.3.1. Folium

Una de las decisiones que hubo que tomar fue qué sistema o librería se utilizaría a la hora de representar gráficamente los datos. Se valoraron opciones software bastante completas como **ArcGIS** pero esta idea se acabó desechándose para desarrollar una aplicación más "a medida", por lo que después de estudiar diferentes opciones se eligió la librería para Python **Folium**[5].

Folium es capaz de generar mapas interactivos en HTML5 y Javascript utilizando la librería *leaflet.js* a partir de objetos Python. Además ofrece gran cantidad de opciones como mapas de calor, callejeros o mapas geográficos y la posibilidad de insertar elementos HTML, por lo que el nivel de personalización de los mapas con esta librería puede ser muy amplio.

Figura 4.4: Visualización del mapa del histórico del año 2007

En este proyecto existen dos funciones que generan mapas:

- Generación de histórico: Esta función recibe como parámetros un rango de fechas y una especie (*ciconia ciconia*, *anser anser* o ambas) y devuelve todos los brotes de gripe aviar detectados entre las fechas, así como los avistamientos de migraciones de la especie seleccionada, además de añadirse una leyenda. Tanto los brotes como las migraciones tienen *pop-ups* con una información ampliada de la que aparece cuando se pasa el ratón por encima.

  Para generar este mapa, cuyo ejemplo puede verse en la Figura 4.4, en el que se muestra el histórico de migraciones y brotes del año 2007, se utilizaron los datos de brotes y migraciones guardados en las colecciones outbreaks y migrations.

- Generación del mapa de alertas: Esta función calcula las alertas y las renderiza en el mapa. El proceso de generación de alertas se detalla en el Capítulo 5. El mapa que

[5] `https://python-visualization.github.io/folium/`

devuelve este módulo incluye los brotes ocurridos en los últimos dos/seis/nueve meses y las alertas que se generarán a partir de los mismos, con una intensidad de color en función de la "probabilidad de la alerta". Además las alertas tienen un *pop-up* que amplían la información geográfica de la misma. La Figura 4.5 incluye una simulación de un mapa de alertas.

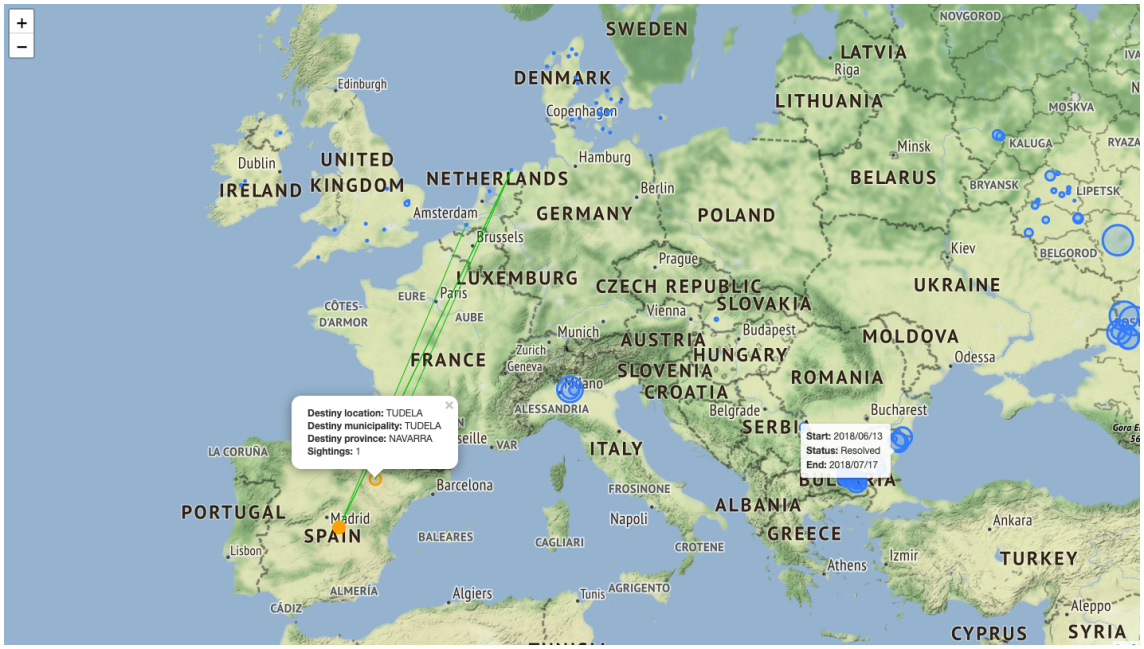

Figura 4.5: Ejemplo de mapa de alertas.

En el Apéndice B, se encuentran tanto la función que calcula y renderiza las alertas.

4.3.2. Bootstrap

Bootstrap[6] es un framework para el desarrollo del frontend en aplicaciones web. Además de contener plantillas de diseño para formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en **HTML** y **CSS**, así como extensiones de JavaScript adicionales, bootstrap ha destacado por la facilidad con la que se pueden crear páginas responsive compatibles con diferentes dispositivos. De esta forma, y como se puede ver en la Figura 4.6, puede abrirse la aplicación tanto desde teléfonos como tablets adaptándose el diseño de la aplicación al tamaño de la pantalla.

La estructura de una página en Bootstrap está basada en un sistema de rejilla, que divide la página en filas y columnas, de forma que el desarrollador establece para cada elemento de la página el ancho del mismo en función del tamaño del dispositivo desde el que se visualiza.

Además se incluyen en la librería otras utilidades como la posibilidad de ocultar los *headers* en pantallas pequeñas o los conocidos como menús **hamburguesa**

[6] `https://getbootstrap.com/`

Figura 4.6: Visualización de la aplicación en diferentes dispositivos

4.3.3. Jinja2

Flask es únicamente un framework para ofrecer servicios basados en HTTP. Para poder publicar contenidos dinámicos en HTML es necesario utilizar **Jinja2**[7].

Jinja2 es un motor de plantillas para Python que permite al desarrollador producir páginas web que contiene código HTML base y marcadores de posisición para que Jinja2 los llene.

Su funcionamiento es muy simple, tienes una plantilla con un montón de "agujeros" en ella y el motor los completa en tiempo de ejecución y devuelve un documento HTML listo para ser enviado al usuario. En el Código 5 vemos un ejemplo extraído de la plantilla que carga el layout de la aplicación en la que caben destacar algunos ejemplos:

- Todo el código en jinja está especificado con llaves, cada vez que se abre un bloque hay que especificar cuando termina.
- Cuando se carga el bloque body, se especifica cual es el formulario que se ha de renderizar. Ésto es porque se tienen dos formularios, uno para la visualización del histórico y otro para la visualización de las alertas. El valor de la variable *form* se especifica en la configuración de las vistas.
- Se aprecia como se pueden insertar fragmentos de otros HTMLs comunes para todas las vistas, como es el caso de los ficheros *nav.html* y *footer.html*

[7] `https://jinja.palletsprojects.com/en/2.10.x/`

```
<div class="wrapper">
    <div class="sidebar" data-color="purple" data-background-color="white">
      <div class="logo">
        <a href="http://informatica.ucm.es" class="simple-text logo-mini">
          <img src="/static/assets/img/logoucm.png" height="120px">
        </a>
      </div>

      {% block sidebar %}{% endblock %}
    </div>
    <div class="main-panel">

      {% block body %}
      {% with form=form  %}
      {% include "nav.html" %}
      {% endwith %}

      <div class="content">
        <div class="container-fluid">
          {% block content %}{% endblock %}
        </div>
      </div>

      {% include "footer.html" %}

      <!-- JavaScript at the bottom for fast page loading -->
      {{ javascript_tag('main_js') | safe }}
      {% block js %}{% endblock %}
      <!-- end scripts -->
      {% endblock %}
    </div>
</div>
```

Código 5: Fragmento del template del layout de la aplicación

Capítulo 5

Sistema de Alertas

"Las cifras no mienten, pero los mentirosos también usan cifras."
— Anónimo

Cabe recordar que el objetivo final del proyecto es desarrollar un sistema que permita predecir casos de Avian Inzluenza en España. Fueron varias las ideas que surgieron para la generación de alertas de posibles brotes de AI, de hecho la versión que se presenta en este proyecto es una primera iteración de un modelo que en un futuro se mejorará añadiendo, por ejemplo, nuevas especies para enriquecer la base de datos de migraciones.

5.1. Generación de alertas

En la Figura 5.1 se pueden observar dos mapas de calor, en la izquierda se ven representados los destinos de aves migratorias en la Península Ibérica, y a la derecha los brotes que se han producido desde el año 2005 en las misma. Se puede observar que existe cierta correlación entre los lugares donde se han producido casos con los sitios preferidos por los animales para estacionar, relación que ya ha sido estudiada por diversos investigadores (Martinez et al., 2008).

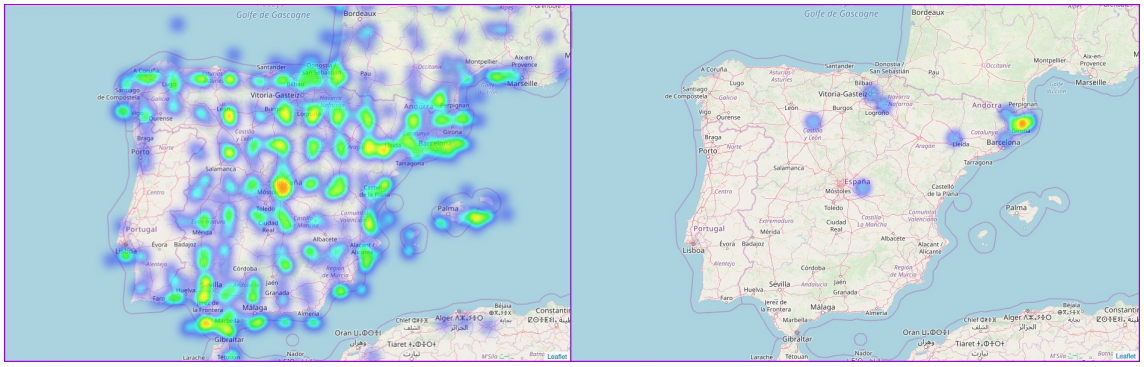

Figura 5.1: Comparativa de mapas de calor de migraciones (izquierda) y brotes (derecha)

Es por esto que se ha optado por utilizar la información de flujos migratorios para la elaboración del sistema. El proceso, mostrado de forma esquemática en la Figura 5.2, que se sigue para generar posibles alarmas es el siguiente:

- Se realiza una búsqueda en la colección *Outbreaks* de los brotes que hayan sido detectados y documentados en Europa en el periodo de tiempo seleccionado(dos, seis o nueve meses).
- De la lista de brotes que devuelve la búsqueda anterior, se selecciona el campo geohash de todos ellos.
- Se consulta al grafo de migraciones por todos los avistamientos cuyo origen coincide en los primeros cuatro dígitos del geohash de los brotes (regiones con una precisión de unos 20km) y su destino está en España.
- En caso de tener resultados en la anterior petición los puntos de destino de esos avistamientos serán lugares con potenciales casos de gripe aviar, pues existe una ruta migratoria desde un punto en el que se ha documentado un caso.

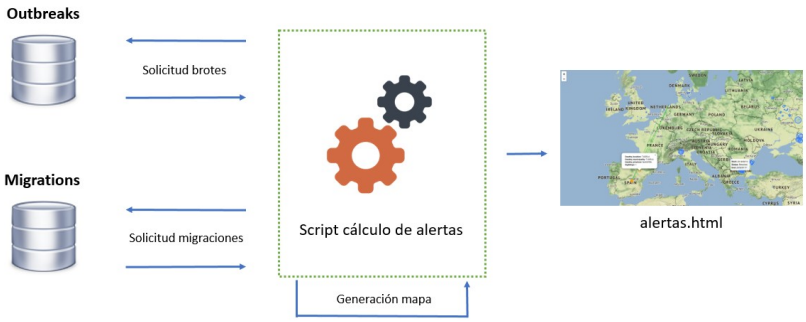

Figura 5.2: Esquema del proceso de generación de alertas

Como veremos en la siguiente sección, este sencillo mecanismo de generación de alarmas resulta efectivo aunque podría ser mejorable añadiendo técnicas de aprendizaje automático, pero a día de hoy no son demasiados fiables fiables debido a:

- Con apenas veinte mil datos de migraciones y unos dos mil datos brotes en Europa, quizás no haya suficientes datos para que las técnicas de **aprendizaje no supervisado** den unos resultados lo suficientemente precisos y más teniendo en cuenta que apenas tenemos cuatro brotes en España con los que validar el modelo resultante.
- Además de la poca cantidad de datos, no existe aún certeza de la correlación migración-brote, por lo que sería un tanto pretencioso darla por hecho y aplicar técnicas de **aprendizaje supervisado**.

Como se ha comentado, un buen trabajo futuro sería enriquecer la base de datos de migraciones con datos de otras organizaciones y asociaciones, así como estudiar la existencia de una relación entre brotes y rutas migratorias para poder mejorar el sistema de alertas utilizando diferentes técnicas de inteligencia artificial.

5.2. Validación del sistema

El método que se ha utilizado para comprobar la efectividad del algoritmo de generación de alertas ha consistido en seleccionar los cuatro brotes documentados en España hasta el día de hoy y, tomando una fotografía de los datos de migraciones (de cigüeñas y gansos) en el momento del caso, comprobar si se hubieran podido predecir con dos, seis y nueve meses de antelación.

De esta forma podemos comprobar si hay o no falsos negativos, es decir si los casos que tenemos documentados hubieran generado alguna alerta. Se van a analizar también caso por caso el número de avisos que se hubieran producido en el resto de la Península, es decir los falsos positivos.

Los falsos positivos, se dan por hecho debido a que las épocas estudiadas han sido de bastante actividad para la AI en Europa, no son preocupantes y de hecho pueden ser necesarios para mantener alerta zonas avícolas en riesgo por ser destino de flujos migratorio.

5.2.1. Brote de Aigüamolls (Gerona)

El primer caso documentado de este brote se produjo el día 17 de febrero de 2017, y terminó afectando a varias granjas de patos, lo que implicó el sacrificio de más de 7000 individuos según datos del portal WAHIS.

Si se dibujan mapas de calor con las hipotéticas alertas que se hubieran generado en esa fecha con los datos de los los dos, seis y nueves anteriores al brote, tal y como puede verse en la Figura 5.3, se observa que se hubieran producido bastantes alertas debido a la intensidad de la enfermedad en Europa.

Figura 5.3: Mapa de calor de alertas producidas en febrero de 2017.

No obstante se puede comprobar que la zona afectada es una "zona caliente". Si hacemos una comprobación en la zona del brote, cuyas regiones de geohash afectadas son *sp98*, *sp9b*, *spd0*, *sp3x*, *sp3z*, *sp6p*, *sp3w*, *sp3y* y *sp6n* podemos observar en las Tabla 5.1, Tabla 5.2 y Tabla 5.3, en las que se muestran los positivos que se hubieran generado con los diferentes rangos de tiempo, que se hubieran generado varias alertas. La columna **Avistamientos** hace referencia al número de aves avistadas a fecha de brote entre el lugar del brote y el lugar de la alerta

Región	Brote origen	Avistamientos
sp3x	*u0tzn3qbgnsr*	1
sp3z	*u0tzn3qbgnsr*	1

Tabla 5.1: Alertas generadas con los datos de dos meses para el brote de 2017 en Gerona

Región	Brote origen	Avistamientos
sp3x	*u0t6q23gdgmg*	2
sp3x	*u0tzn3qbgnsr*	1
sp3x	*u0qhtmcq9whs*	1
sp3x	*u0qhtzz0uwq0*	1
sp6n	*u0w94jc0f8fc*	1
sp3z	*u0tzn3qbgnsr*	1
sp3z	*u0mhheb8phy8*	1
sp3z	*u0mhh7c2nhy0*	1

Tabla 5.2: Alertas generadas con los datos de seis meses para el brote de 2017 en Gerona

5.2.2. Brote de Navas de Fuentes (Palencia)

Este brote se documentó el 3 de enero de 2017, cuando se encontraron dos patos muertos en una laguna. Al contrario que el caso anterior, apenas tuvo incidencia mediática ni impacto económico.

En este caso el mapa de calor con falsos positivos generado para las tres ventanas de tiempo es el mismo, representado en la Figura 5.4

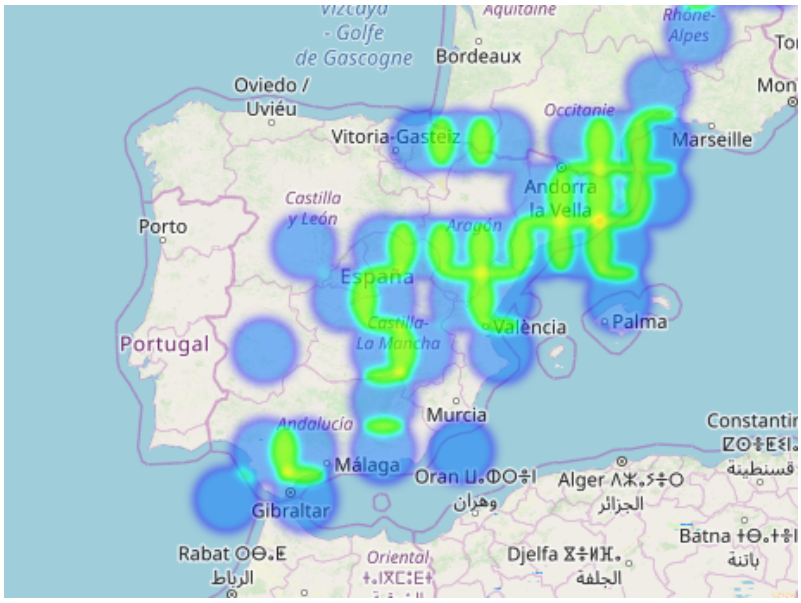

Figura 5.4: Mapa de calor de alertas producidas en enero de 2017.

Para este brote no se habría producido ninguna alerta en la zona afectada. Seguramente esto fuese debido a que los individuos afectados fueron patos y por el momento el sistema de alertas trabaja con avistamientos de gansos y cigüeñas.

Región	Brote origen	Avistamientos
sp3x	*u0t6q23gdgmg*	2
sp3x	*u0tzn3qbgnsr*	1
sp3x	*u0qhtmcq9whs*	1
sp3x	*u0qhtzz0uwq0*	1
sp6n	*u0w94jc0f8fc*	1
sp3z	*u0tzn3qbgnsr*	1
sp3z	*u0mhheb8phy8*	1
sp3z	*u0mhh7c2nhy0*	1

Tabla 5.3: Alertas generadas con los datos de nueve meses para el brote de 2017 en Gerona

5.2.3. Brote de Salburua (Vitoria)

En mayo de 2006 fue encontrado un somormujo lavanco (*Podiceps cristatus*) muerto en un humedal de Salburua. Se trató del primer caso de gripe aviar registrado en España, teniendo una gran transcendencia mediática. No obstante, aunque se realizaron pruebas a más aves ninguna dio resultado positivo[1].

Al ser de los primeros brotes, en este caso no se habrían generado tantas alertas como en los anteriores. Con una ventana de tiempo de 60 días antes del brote no se habría generado ningún falso positivo en España. Para las ventanas de tiempo de seis y nueve meses sin embargo si se hubieran generado alertas, representadas mediante el mapa de la Figura 5.5

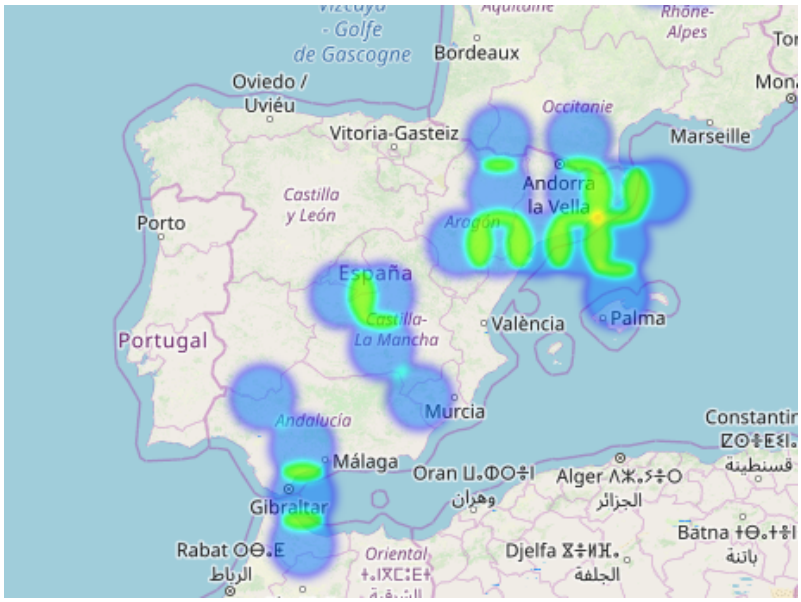

Figura 5.5: Alertas producidas en mayo de 2006 con los datos de los seis y nueve meses previos.

Para este brote tampoco se hubieran generado alertas en la región afectada.

[1] `https://elpais.com/diario/2006/07/12/sociedad/1152655208_850215.html`

5.2.4. Brote de Almoguera (Guadalajara)

Este brote se produjo en octubre de 2009 y fue de los pocos registrados en la Unión Europea durante ese año. Obligó al sacrificio de más de 300000 gallinas en una granja de Almoguera. Este caso es interesante para este estudio puesto que el dueño de la explotación declaró que el brote pudo originarse a través de una bandada de cigüeñas que semanas antes se asentaron cerca de la explotación[2].

Sin embargo, como ya se ha comentado no se generaron apenas casos de gripe aviar en Europa durante ese año, por lo que el sistema de alertas queda también invalidado para este caso. No obstante podría ampliarse para el estudio con los casos de brotes de África, pues se conoce que algunos países de este continente forman parte del flujo migratorio de estas aves. Además, como pudo verse en la Figura 5.1 las regiones afectadas en este caso son de los puntos que más aves reciben de la península.

[2] https://www.lavozdegalicia.es/noticia/sociedad/2009/10/17/dueno-granja-guadalajara-afectada-gripe-aviar-0003_8042512.htm

Capítulo 6

Conclusiones y Trabajo Futuro

"He aprendido que los errores pueden ser tan buenos profesores como el éxito."
— Jack Welch JR, Empresario y escritor.

Llegados a este punto es el momento de hacer una retrospectiva y analizar las conclusiones obtenidas, así como las futuras líneas de trabajo para continuar desarrollando la aplicación.

6.1. Conclusiones

En este proyecto se ha desarrollado un sistema de alertas que intenta predecir casos de Avian Influenza en Europa, utilizando el histórico de casos documentados adquiridos mediante el método de **web Scrapping** del portal WAHIS-OIE así como datos migratorios proporcionados por la Sociedad Española de Ornitología obtenidos mediante técnicas de anillamiento.

Para ello, se han utilizado diversas tecnologías a distintos niveles, homogeneizadas en una aplicación accesible desde un navegador web, programado en **Python** y publicado gracias al **framework Flask**.

Para cotejar el correcto funcionamiento del módulo de alertas, se han comprobado los hipotéticos resultados que se hubieran obtenido en los cuatro casos de gripe aviar documentados en España en el momento en que se produjo el brote. Para ello se han cargado los datos de brotes y migraciones disponibles en el instante del caso y se han aplicado ventanas de tiempo de los dos, seis y nueve meses previos a la infección.

Los resultados obtenidos en este proceso no han sido del todo satisfactorios debidos a diferentes motivos:

- Sólo se han estudiado datos migratorios obtenidos por el método de anillamiento de las especies *Ciconia ciconia* (cigüeña) y *Anser anser* (gansos), por lo que brotes provocados por otras especies como los casos de Salburua y Navas de Fuentes no podrían haber sido predichos.

- Se parte del histórico de brotes documentados en Europa, aunque se conoce que muchos de los animales migratorios que estacionan en España proceden de África. Quizás este sea el principal motivo por el cual no se habría detectado de forma temprana el caso de Almoguera.

Sin embargo, para el brote de Aigüamolls sí que se podría haber mantenido un estado de aviso en las regiones afectadas con la información de los meses previos, pudiéndose minimizar los efectos de la infección.

Pese a no ser muy buenos resultados, siendo una primera iteración del módulo, se han encontrado varías vías de desarrollo para mejorar la herramienta que podrían incidir positivamente en la efectividad del sistema, como ampliar el número de especies a estudiar o ampliar la base de datos de migraciones.

6.2. Trabajo Futuro

Como se ha ido comentando a lo largo del documento, existen varios puntos de mejora para seguir desarrollando el proyecto. Quizás el más inmediato es mejorar el proceso de generación de alertas, automatizándolo todas las noches a la vez que se actualiza la información de brotes, incluso se podría notificar mediante correo electrónico u otros medios en caso de surgir alarmas nuevas.

También en el ámbito de las alertas hay otros aspectos a mejorar. El primero, y posiblemente más prioritario, sería empezar a utilizar datos de avistamientos de otras especies y conseguir más datos de migraciones para "afinar" el proceso de cálculo. Además con una base de datos mayor sería posible empezar a plantearse aplicar modelos de aprendizaje no supervisado.

Una vez sea estable el sistema de alertas para España el siguiente paso sería implementar un sistema de alertas a nivel Europeo, partiendo de los brotes y datos migratorios de Asia. Con vistas a esto, los primeros ya se están descargando e introduciendo en la base de datos de forma periódica.

En el entorno de las bases de datos, hay margen de mejora en las consultas a Neo4j si se utiliza el sistema de querys que proporciona el OGM de **py2neo** en vez de utilizar Cypher, puesto que las primeras ya han sido optimizadas previamente tanto en la query como en la serialización del resultado.

Otra utilidad que podría ser beneficiosa para el usuario final sería la de permitir importar datos desde un fichero de texto CSV para facilitar la ingestión de datos migratorios, además se podría desarrollar adicionalmente la posibilidad de exportar datos a diferentes formatos como podrían ser PDF o CSV así como descargar los mapas como imágenes.

Por último, con el uso aparecerán pequeños bugs o fallos de frontend, como textos cortados en determinados tamaños de pantalla o iconos que no se muestran en todos los entornos. Se espera solucionar estos fallos en el futuro a medida que vayan manifestándose.

Chapter 7

Introduction

"Data are the new science. Data Science is the answer "
— Pat Gelsinger, VMWare CEO

Avian influenza (AI) is a worldwide disease that affects both domestic and wild birds. Migratory bird movements (especially aquatic birds) have played a very important role in intercontinental diffusion of the most prevalent subtypes currently in Europe (H5N1 and H5N8), and transmission between wildlife-domestic interface is recurrent in spite of the measures of biosecurity and control.

Epidemiology is considering as fundamental science for preventive medicine and is applied in veterinary science to investigate the dynamics and factors associated with diseases which affect animal populations. The current focus of epidemiology is no longer the animal's individual treatment; as it happened until the Fifties, nor the improvements in health and productivity of the herds as it was done until the Nineties. Currently, the focus whole food chain and standardization of the food quality and safety within a global context that also includes human health and wildlife, known as ***One Health***. It has become evident that historical separation of surveillance of animal and people diseases ignores the inherent connection of biological systems, making it necessary for *One health* approach. This global vision of health requires more extensive and multidisciplinary scientific studies as is the link between computer science and data science with health specialists, as it is intended to show in the development of this alert system for the ***highly pathogenic avian influenza (IAAP)*** early detection in Spain.

7.1. Motivation

The AI epidemic that were located in Europe the last 2006 with economic consequences unwelcome of the poultry sector. After a calm couple of years, since 2016, there has been a new outbreaks wave both in wild and in domestic birds placed in Europe with more than thirty affected countries, including Spain. Spain was affected in 2017 with 2 wild birds' outbreaks in Palencia and Girona and an outbreak in domestic birds in Girona, one of the most important poultry areas of the country.

Spain receives millions of birds each year they receive from different locations in Europe, as you can see in Figure 7.1. These migrations are a constant threat to a constant AI

introduction, which represents an important risk for the poultry sector and, in lesser extent, for the public health of our country.

It should be noted that in Spain, the final agricultural production constitutes 4% of the G.D.P., from which 38% is represented by the Final Livestock Production with more than sixteen thousand millions of euros. They emphasize the poultry sector that occupies the second best in meat production in Spain and in the EU only behind the United Kingdom.

The importance of animal health in Spain, due to its impact on livestock production, derives from the fact that it is an important pillar of the agri-food industry and one of the main economic activities of our country, with a positive external trade balance and an increasing trend in recent years. For this reason, it is necessary to keep livestock populations free of diseases that could pose an economic risk to their stability and profitability, thus avoiding possible economic restrictions and market disturbances.

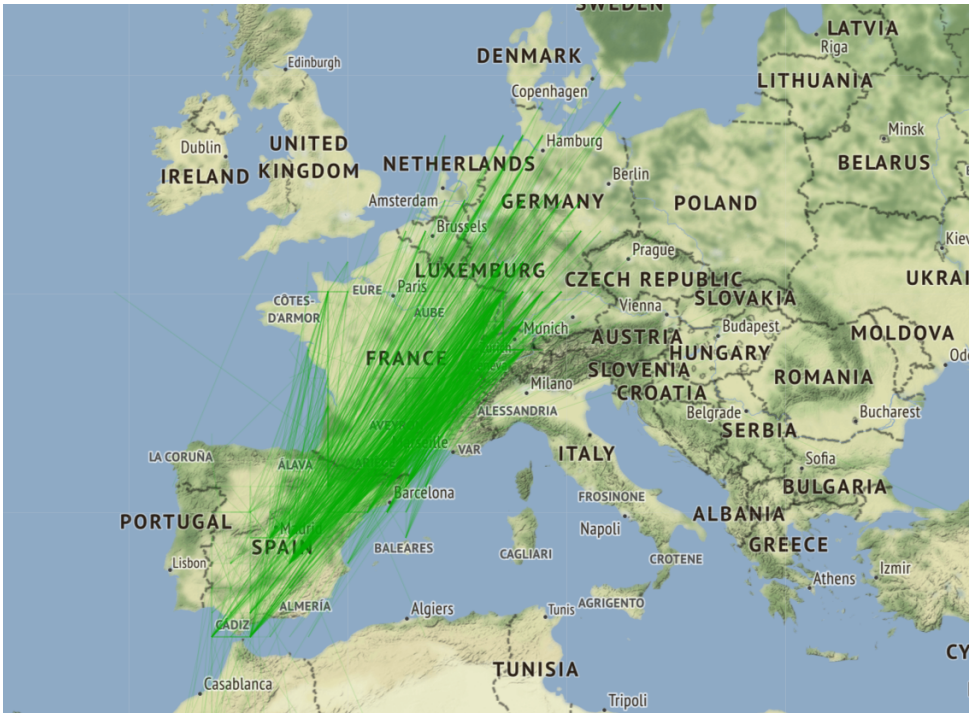

Figure 7.1: Migratory sightings of *Ciconia ciconia* (common stork) from Europe to Spain

7.2. Objectives

The main objective of the project is to develop a tool able to predict, with a high level of reliability, Avian Influenza alerts in Spain from outbreaks documented in Europe. This requires:

- An automatically updated data source that allows acquisition and exposure of information on outbreaks and migrations in a simple way.
- A storage efficient model, both in disk size and in yield, to information from outbreaks and migrations.

- A graphical application to visualize the notifications. To allow greater access it is recommendable the hosted on a web server.
- A simple and intuitive interface, due to application must be used by a very wide range of users.
- A compatible interface with different screen sizes, so it can be used both in computers, as in smartphones and in tablets.

7.3. Workplan

The work plan to the project implementation should follow the next phases:

- **Problem Analysis**. At this stage it is about understanding the problem. To this aim the different data sources to do this project and potential users requirements are to be analysed.
- **Software architecture design**. At this time, we choose the modules in which we have divided the application, as well as the necessary technologies for the development of each one of them.
- **Study of the technologies to use**. Once the architecture to be developed is planned, the technologies needed to do so will have to be studied and configured.
- **Software development**. At this point, all the second phase modules resulted will be implemented and joined.
- **Evaluation of the tool**. Thanks to the developed software it would be time to test the application's usability and especially the alert system implemented. To do this, the Avian Influenza outbreaks documented in Spain will be used until the date to verify how the tool would have behaved.
- **Drafting of the report and documentation of the project**. Finally, all the information acquired during the development will be collected for the drafting of the documentation.

7.4. Document structure

The rest of the document has been divided into chapters following this structure:

- The chapter 2 addresses the current problem status in several approaches, since data acquisition in migratory birds' specific field, to cross-platform applications design, go through libraries and databases for large amounts of data study.
- The chapter 3 involves the different processes that it has been followed to data acquire, as well as the storage; employing the different technologies used.
- Once the data is acquired, the next step is to explain the use made of the same, which is detailed in chapter 4.

- The chapter 5 explains the different steps followed as well as the tools applied to develop the application that this project is based on.
- Finally, chapter 6 analyses the obtained conclusions during and after the development. Furthermore, on this chapter are indicated the following steps or guidelines to continue developing the project

Chapter 8

Conclusions and Future Work

"I have learned that mistakes can be as good teachers as success."
— Jack Welch JR, Businessman and writer.

At this point it is time to look back and analyse the conclusions obtained, as well as future work lines to continue developing the application.

8.1. Conclusions

In this project an alert system that tries to predict cases of Avian Influenza (AI) in Europe has been developed. For this purpose, history of documented cases has been used and It has been acquired using **web Scrapping** method by WAHIS-OIE portal, as well as migratory data provided by the Spanish Ornithological Society using bird banding techniques.

To that end, various technologies have been used at different levels, an accessible application enables to homogenize it from a web browser. This web browser is programmed in **Python** and published thanks to **Flask framework**.

To collate the proper function of the alerts module, the hypothetical results obtained in the four cases of avian influenza documented in Spain have been checked to the time with the outbreak took place. Outbreak data and migrations available at the time of the case have been loaded, and time windows have been applied of the two, six and nine months previous to infection.

The results obtained in this process have not been entirely satisfactory due to different reasons:

- Only have been studied migratory data obtained by bird banding method of the *Ciconia ciconia* (stork) and *Anser anser* (geese) species, so other outbreaks caused by different species could not have been predicted (such as the cases of Salburua and Navas de Fuentes).
- It is based on outbreak documented history placed in Europe, although it is known that many of the migratory animal come to Spain from Africa. Perhaps this is the main reason why the Almoguera's case could not have been detected.

However, for Aigüamolls outbreak could has been maintained an early-warning status on affected regions thanks to previous months' information and might it has been able to minimize infection effects.

In spite of results are not very good, being a first module iteration, we have found several ways for development to improve the tool that could has a positively influence on system effectiveness, such as expanding the number of species to study or expand the migration database.

8.2. Future work

As has been commented throughout the document, there are several points for improvement in project development. Maybe the most immediate improvement point is generation alerts process, it could achieve automating each night while outbreaks information is updating, it could may allow been notified by email or other ways when new warnings appears.

Also in alerts field there are other aspects to progress. First step, and probably the most priority, could be start using other species sightings data and obtaining more migration data in order to "refine" calculation process. Furthermore, with a larger database, it would be possible to start considering to apply learning models not supervised.

Once alerts system in Spain is stable, the next step would be to implement a warning at European level, based on outbreaks and migratory data from Asia. With this aim in mind, the first data are already being downloaded and introduced into the database regularly.

In database environment, there is room for an improvement in query with Neo4j if you would use the query system provided by **py2neo's GMO** instead of using Cypher, because of the firsts have already been optimized previously both in query as in result serialization.

Another utility that could be beneficial for the end user would be to allow data import from a CSV text file to facilitate migratory data ingestion. In addition, the possibility of data export to different formats such as PDF or CSV as well as download maps as images.

Finally, with the use will appear small bugs or frontend faults, as cut texts into certain screen sizes or icons that are not shown in all environments. It is expected to solve these failures in the future as they manifest themselves.

Bibliografía

La lectura es una conversación con los hombres
más ilustres de los siglos pasados.

René Descartes

Bridge, E. S., Thorup, K., Bowlin, M. S., Chilson, P. B., Diehl, R. H., Fléron, R. W., Hartl, P., Kays, R., Kelly, J. F., Robinson, W. D. y Wikelski, M. Technology on the move: Recent and forthcoming innovations for tracking migratory birds. *BioScience*, vol. 61(9), Disponible en `https://academic.oup.com/bioscience/article/61/9/689/395051`.

Fiedler, W. New technologies for monitoring bird migration and behaviour. *Ringing & Migration*, vol. 24(3), Disponible en `https://doi.org/10.1080/03078698.2009.9674389`.

González, G. G. La influenza aviar. insistencia mediática, alarma social y efectos socio-económicos. *Real Academia Nacional de Farmacia*, Disponible en `https://www.analesranf.com/index.php/mono/article/view/594/611`.

Lin, T., Winner, K., Bernstein, G., Mittal, A., Dokter, A. M., Horton, K. G., Nilsson, C., Doren, B. M. V., Farnsworth, A., Sorte, F. A. L., Maji, S. y Sheldon, D. Mistnet: Measuring historical bird migration in the us using archived weather radar data and convolutional neural networks. *British Ecological Society*, Disponible en `https://besjournals.onlinelibrary.wiley.com/doi/epdf/10.1111/2041-210X.13280`.

Martinez, M., Perez, A. M., de la Torre, A., Iglesias, I. y Muñoz, M. J. Association between number of wild birds sampled for identification of h5n1 avian influenza virus and incidence of the disease in the european union. Disponible en `https://doi.org/10.1111/j.1865-1682.2008.01046.x`.

Naef-Daenzer, B., Früh, D., Stalder, M., Wetli, P. y Weise, E. Miniaturization (0.2 g) and evaluation of attachment techniques of telemetry transmitters. *Journal of Experimental Biology*, vol. 208(9), Disponible en `https://jeb.biologists.org/content/208/21/4063`.

Okada, H., Itoh, T., Suzuki, K. y Tsukamoto, K. Wireless sensor system for detection of avian influenza outbreak farms at an early stage. páginas 1374 – 1377. 2009.

Su, J.-H., Piao, Y.-C. y Yan, Z. L. . B.-P. Modeling habitat suitability of migratory birds from remote sensing images using convolutional neural networks. *Animals*, vol. 8(5), Disponible en https://doi.org/10.3390/ani8050066.

Swatantran, A., Dubayah, R., Goetz, S., Hofton, M., Betts, M. G., Sun, M., Simard, M. y Holmes, R. Mapping migratory bird prevalence using remote sensing data fusion. *PLoS ONE*, vol. 7(1), Disponible en https://doi.org/10.1371/journal.pone.0028922.

Apéndice A

Web scraping

En este apéndice se adjunta y explica el contenido del script necesario para el proceso de web scraping.

Función principal donde se comienza el proceso:

```
1  import requests
2  import re
3
4  import pymongo
5  from pymongo import MongoClient
6  from datetime import datetime
7
8  #GLOBALS
9
10 client= MongoClient('mongodb://localhost:27017/')
11 db = client.lv
12 outbreaks = db.outbreaks
13
14 def main(argv):
15         # 15 - Highli Path Avian influenza
16         # 201 - Lowi Path Avian influenza
17         # 1164 - Highly pathogenic influenza A viruses (infection with)
18         #Enfermedades a scrapear
19         diseases = ['15', '201', '1164']
20         global lista
21         global disease;
22     #Lista de países asiáticos
23         asian_countries = ['AFG', 'ARM', 'AZE', 'BHR', 'BGD', 'BTN', 'CHN',
24                         'CYP', 'KHM', 'TWN', 'GEO', 'HKG', 'IND', 'IDN',
```

```
25                      'IRN', 'IRQ', 'ISR', 'JPN', 'JOR', 'KAZ', 'KOR',
26                      'KWT', 'KGZ', 'LAO', 'LBN', 'MYS', 'MNG', 'MDV',
27                      'MMR', 'PAK', 'NPL', 'OMN', 'PRK', 'PSE', 'PHL',
28                      'QAT', 'RUS', 'SAU', 'SGP', 'LKA', 'SYR', 'TJK',
29                      'TLS', 'THA', 'TUR', 'TKM', 'ARE', 'UZB', 'VNM',
30                      'YEM']
31      #Lista de países europeos
32          european_countries = ['ALB', 'DEU', 'AND', 'AUT', 'BEL', 'BLR', 'BIH',
33                            'BGR', 'CYP', 'HRV', 'DNK', 'SVK', 'SVN', 'ESP',
34                            'EST', 'FIN', 'FRA', 'GRC', 'HUN', 'IRL', 'ISL',
35                            'ITA', 'LVA', 'LIE', 'LTU', 'LUX', 'MKD', 'MLT',
36                            'MDA', 'MCO', 'MNE', 'NOR', 'NLD', 'POL', 'PRT',
37                            'GBR', 'CZE', 'ROU', 'RUS', 'SMR', 'SRB', 'SWE',
38                            'CHE', 'UKR', 'VAT']
39
40      # Url donde comienza el proceso de scraping
41          url= 'http://www.oie.int/wahis_2/public/wahid.php/Diseaseinformation/Immsummary'
42
43      # Años de búsqueda
44          start_year = 2005
45          finish_year = 2019
46      # Algunos parámetros que hay que hardcodear al formulario
47          disease_id_hidden = 1 # FMD
48
49          for disease_id in diseases:
50                  oblist = []
51                  disease = disease_id
52                  counter = 0
53                  for year in range(start_year, finish_year+1):
54              #Se va creando una lista con los brotes a los que acceder dentro de la página
55                          r = requests.post(url, data = {'year':year,
56                          'disease_type_hidden':1,
57                                'disease_id_hidden':disease_id})
58              # Se obtienen tuplas con el formato (País, Código, ID) para cada brote
59                          p = re.compile("""outbreak_country\">[ \t\r\n]
60                          +([A-Za-z \-.']+)[^(]*\('([A-Z]{3})',([0-9]+)\);"""
61                             ,re.DOTALL & re.MULTILINE)
62                          m = p.findall(r.content.decode('latin1'))
63                          oblist = oblist + m
64
65          # Para cada elemento de la lista anterior
```

```
66                  for obs in oblist:
67                          cty, code, id = obs
68                          print("Outbreak {} of {}".format(counter, len(oblist)))
69                          print("\rGetting Outbreaks in {}".format(cty))
70                          if(code in asian_countries) or (code in european_countries):
71                                  print("Getting Outbreaks in {}".format(cty))
72                  # Se accede a otra página con la información del brote
73                                  get_cty_obs(code,id, disease, year)
74                          counter += 1
```

La función *main* llama a la siguiente, que recibe un código de país, un código de enfermedad y un año y busca todos los casos documentados de esa enfermedad:

```
1  def get_cty_obs(code,id,disease, year):
2          global ob_ids
3          global db
4          global outbreaks
5
6      #Se obtiene la lista de brotes con los parámetros pasados
7          url ="""
8  http://www.oie.int/wahis_2/public/wahid.php/Diseaseinformation/Immsummary/listoutbreak
9      """
10
11         r = requests.post(url, data = {'reportid':id, 'summary_country':code})
12         p = re.compile('outbreak_report\("([A-Z]{3})",([0-9]+)\)',
13                    re.DOTALL & re.MULTILINE * re.IGNORECASE)
14         ob_list = p.findall(r.content.decode('latin1'))
15         print('Getting data for outbreak of disease {} in country {}\n'.format(id,
16                                                                              code))
17         total = len(ob_list)
18         count = 1
19         #Se recorre la lista, si el brote ya está en la base de datos se ignora,
20         # y si no está se accede a otra función que scrapea la información del brote
21         for ob in ob_list:
22                 cty,id = ob
23                 #miramos si está en la bbdd y si no está la leemos
24                 if outbreaks.find({'oieid':id}).count() == 0:
25                         print("Getting data of (ID {})          ".format(id))
26                         #función que scrapea la información de un brote concreto
27                         get_ob_page(cty,id, disease, year)
28                 else:
29                         print ("ID {} Ignored, already in database".format(id))
```

Por último, esta es la función que accede a la página con el informe de un brote concreto y extrae toda la información, la cual se inserta en un diccionario que es ingestado en la colección outbeaks de MongoDB.

```
def get_ob_page(cty,id, disease, year):
        global outbreaks
        url ="""
http://www.oie.int/wahis_2/public/wahid.php/Diseaseinformation/Immsummary/outbreakreport
"""
    # Se accede a la url del reporte del brote en concreto y se extrae la información
        r = requests.post(url, data = {'reportid':id, 'summary_country':cty})
    page = r.content.decode('latin1')
    ob = ''
    p = re.compile('start of the event</td>[^>]+>(\d{1,2}/\d{1,2}/\d{4}).*?'
                   'Outbreak Status</td>[^>]+>(\w+).*?'
                   'resolution of the outbreak</td>[^>]+>(\d{1,2}/\d{1,2}/\d{4})?.*?'
                   'ta_left">[^<]+</td>[^>]+>(\w+)?.*?'
                   'ta_left">[^<]+</td>[^>]+>(\w+)?.*?'
                   'ta_left">[^<]+</td>[^>]+>(\w+)?.*?'
                   'Unit Type</td>[^>]+>(\w+).*?'
                   'Location</td>[^>]+>(\w+).*?'
                   'Latitude</td>[^>]+>(-?\d+\.\d+).*?'
                   'Longitude</td>[^>]+>(-?\d+\.\d+).*?'
                   'Description of Affected Population</td>[^>]+>(.*?)</td>'
                   ,re.DOTALL & re.MULTILINE * re.IGNORECASE)

    anlist = p.findall(page)
    if len(anlist) > 0:
        ob = anlist[0]
    else:
        ob = ('','','','','','','','','','', '')
    p = re.compile('vacborder">([^<]*)?</td>.*?'
                'vacborder">(\d+)?</td>.*?'
                'vacborder">(\d+)?</td>.*?'
                'vacborder">(\d+)?</td>.*?'
                'vacborder(?: last)?">(\d+)?</td>.*?',
                    re.DOTALL & re.MULTILINE * re.IGNORECASE)
    anlist = p.findall(page)

    # Se descartan los brotes que están sin fecha.
        if ob[2] != "" and ob[0] != "":
                end = datetime.strptime(ob[2], "%d/%m/%Y")
```

```
39                  start = datetime.strptime(ob[0], "%d/%m/%Y")
40          # Se construye el diccionario
41                  outbreak = {}
42                  outbreak["oieid"] = id
43                  outbreak["diseade_id"] = disease
44                  outbreak["country"] = cty
45                  outbreak["start"] = start
46                  outbreak["status"] = ob[1]
47                  outbreak["end"] = end
48                  outbreak["city"] = ob[3]
49                  outbreak["district"] = ob[4]
50                  outbreak["subdistrict"] = ob[5]
51                  outbreak["epiunit"] = ob[6]
52                  outbreak["location"] = ob[7]
53                  outbreak["lat"] = ob[8]
54                  outbreak["long"] = ob[9]
55                  outbreak["affected_population"] = ob[10]
56                  if len(anlist)>0:
57                          outbreak["species"] = anlist[0][0]
58                          outbreak["at_risk"] = anlist[0][1]
59                          outbreak["cases"] = anlist[0][2]
60                          outbreak["deaths"] = anlist[0][3]
61                          outbreak["preventive_killed"] = anlist[0][4]
62                  else:
63                          outbreak["species"] = ""
64                          outbreak["at_risk"] = ""
65                          outbreak["cases"] = ""
66                          outbreak["deaths"] = ""
67                          outbreak["preventive_killed"] = ""
68
69                  print ("Writing {} in database...".format(id))
70                  #Y se inserta en la base de datos
71                  outbreaks.insert_one(outbreak)
```

Apéndice B

Scripts Utilizados

Se adjuntan a continuación una serie de funciones útiles para el desarrollo de la herramienta que se han mencionado a lo largo de la memoria.

B.1. Tratamiento del CSV de Migraciones

Este script se encarga del procesamiento del csv de migraciones y su inserción en MongoDB. Para ello se vale de las diferente funciones facilitadas por librería **Pandas**

```
1  import pandas as pd
2  import pymongo
3  from pymongo import MongoClient
4  import json
5
6  file = "migrations.csv"
7
8  df = pd.read_csv(file, sep=';', encoding='latin-1', low_memory = False)
9
10 #Ahora borramos todos los que tengan 'localidad confidencial'
11 df = df[df.Localidad != 'Localidad confidencial']
12 df = df[df.LocalidadR != 'Localidad confidencial']
13
14 #Ahora borramos las columnas que nos sobran:
15 df.drop(['Cod_Localidad', 'Cod_LocalidadR', 'EspecieR',
16 'Verificacion', 'InfoAnillaR', 'AmpliacionAnillaR', 'CentroR'], axis='columns',
17 inplace=True)
18
19 #Ahora vamos a dar formato a las coordenadas:
20 df['CuadranteLongitudR'] = df['CuadranteLongitudR'].map({'W':'-', 'E':'+'})
```

```
df['CuadranteLongitud'] = df['CuadranteLongitud'].map({'W':'-', 'E':'+'})
df['CuadranteLatitud'] = df['CuadranteLatitud'].map({'S':'-', 'N':'+'})
df['CuadranteLatitudR'] = df['CuadranteLatitudR'].map({'S':'-', 'N':'+'})

df['Lat'] = df['CuadranteLatitud'].map(str) + df['LatitudGrados'].map(str) + \
'.' + df['LatitudMinutos'].map(str)
df['Long'] = df ['CuadranteLongitud'].map(str) + df['LongitudGrados'].map(str) + \
'.' + df['LongitudMinutos'].map(str)

df['LatR'] = df['CuadranteLatitudR'].map(str) + df['LatitudGradosR'].map(str) + \
'.' + df['LatitudMinutosR'].map(str)
df['LongR'] = df ['CuadranteLongitudR'].map(str) + df['LongitudGradosR'].map(str) + \
'.' + df['LongitudMinutosR'].map(str)

#Con las coordenadas formateadas, se da formato a las fechas.
df['FechaAnillamiento'] = pd.to_datetime(df['FechaAnillamiento'], format= "%d/%m/%y")
df['FechaRecuperacion'] = pd.to_datetime(df['FechaRecuperacion'], format= "%d/%m/%y")

#Se eliminan el resto de columnas que ya se han utilizado y no se quieren guardar
df.drop(['LatitudGrados', 'LatitudMinutos', 'LongitudGrados', 'LongitudMinutos',
'CuadranteLatitud', 'CuadranteLongitud'], axis='columns', inplace=True)
df.drop(['LatitudGradosR', 'LatitudMinutosR', 'LongitudGradosR', 'LongitudMinutosR',
'CuadranteLatitudR', 'CuadranteLongitudR'],axis='columns', inplace=True)
df.drop(['Unnamed: 31'], axis='columns', inplace=True)

#Se configura la conexión con la BBDD...
client= MongoClient('mongodb://localhost:27017/')
db = client.lv
migrations = db.migrations
records = df.to_dict(orient='records')

#... y por ultimo guardamos la información en bulk en la colección
migrations.insert_many(records)
```

B.2. Generación Alertas

A continuación se muestra el proceso por el cual se generan las alertas, resumiendo se trata de una función que recibe un número de días, que son proporcionados desde el formulario del front-end y pueden ser dos meses, seis meses o nueve meses.

Esta función cruza los brotes de ese rango de fecha con los movimientos migratorios del grafo en neo4j para dibujar las alertas en España

```
import geohash
import pymongo
from pymongo import MongoClient
import folium
from datetime import datetime
from datetime import timedelta
from py2neo import Graph
import math

#Se configuran las conexciones con la base de datos
graph = Graph("bolt://localhost:7687", auth=("neo4j", "ed4r;bnf"))
client= MongoClient('mongodb://localhost:27017/')
db = client.lv
outbreaks = db.outbreaks
migrations = db.migrations

#Se seleccionan los países a filtrar para extraer los brotes, en este caso los europeos
european_countries = ['ALB', 'DEU', 'AND', 'AUT', 'BEL', 'BLR', 'BIH', 'BGR',
                      'CYP', 'HRV', 'DNK', 'SVK', 'SVN', 'ESP', 'EST', 'FIN',
                      'FRA', 'GRC', 'HUN', 'IRL', 'ISL', 'ITA', 'LVA', 'LIE',
                      'LTU', 'LUX', 'MKD', 'MLT', 'MDA', 'MCO', 'MNE', 'NOR',
                      'NLD', 'POL', 'PRT', 'GBR', 'CZE', 'ROU', 'RUS', 'SMR',
                      'SRB', 'SWE', 'CHE', 'UKR', 'VAT']


def map_alerts(days):

    mapdiseases = {
        '15': "Highli Path Avian influenza",
        '201': "Lowi Path Avian influenza",
        '1164': "Highly pathogenic influenza A viruses (infection with)",
    }

    #Se crea un objeto de tipo mapa
    map = folium.Map(location=[47.9600202, 9.9153559],
                     tiles='Stamen Terrain',
                     zoom_start=5)

    #Se configuran diferentes colores para las migracioens de diferentes especies
    mapcolors={
       'MIGRA1340': '#04B404',
```

```
42          'MIGRA1610': '#FE2E64',
43      }
44
45      #Se obtiene el rango de fechas
46      end = datetime.now()
47      delta = timedelta(days = int(days))
48      start = end - delta
49
50      for country in european_countries:
51          data_outbreaks = outbreaks.find({"country": country,
52                                           "start":{"$gte": start, "$lte":end }})
53
54          # Para cada brote se crea su objeto en el mapa
55          for data in data_outbreaks:
56              #Lo primero es comprobar si tiene información suficiente para representarlo...
57              if data['lat'] == None or data['end'] == None or data['start'] == None:
58                  continue
59              if data['cases'] == "" or data['cases' ]== "0":
60                  if data['deaths'] == "" or data['deaths'] == '0':
61                      continue
62                  try:
63                      radio = math.log(int(data['deaths'])) * 5000
64                  except:
65                      print("No se puede representar este brote: {}".format(data))
66              else:
67                  try:
68                      radio = math.log(int(data['cases'])) * 5000
69                  except:
70                      print("No se puede representar este brote: {}".format(data))
71
72              #... Se añade el resto de información...
73              start_date = data['start'].strftime("%Y/%m/%d")
74              end_date = data['end'].strftime("%Y/%m/%d")
75              message="""<b>Start:</b> {} <br> <b>Status:</b> {} <br>
76              <b>End:</b> {}""".format(start_date, data['status'],
77                      end_date)
78              popup_message = """
79              <p><b>Disease:</b> {}<br>
80              <b>Start:</b> {}<br>
81              <b>End:</b> {}<br>
82              <b>Status:</b> {}<br>
```

```
83              <b>Cases:</b> {}<br>
84              <b>Deaths:</b> {}<br>
85              <b>More Information:</b> {}<br>
86              </p>
87              """.format(mapdiseases[data["diseade_id"]], data["start"], data["end"],
88                       data["status"], data["cases"], data["deaths"],
89                       data["affected_population"])
90              #... Y se añade al mapa
91              folium.Circle(
92                  location=[float(data['lat']), float(data['long'])],
93                  radius = radio,
94                  fill = True,
95                  popup=folium.Popup(popup_message, max_width=400),
96                  tooltip=message
97                  ).add_to(map)
98
99              #Ahora se comprueba en el grafo si existen rutas desde la región del brote...
100             #... Para ello se crea la query ...
101             query = "MATCH p=(a:Region{location:"
102             query += "'{}'".format(data['geohash'][:4])
103             query += "})-[r]->(b) "
104             query += "RETURN r.valor, b.location, b.localidad, "
105             query += "b.municipio, b.provincia, type(r)"
106             #... Y se ejecuta en Neo4j ...
107             migrations = graph.run(query).data()
108             origin_coordinates_f = [0, 0]
109             destiny_coordinates_f = [0, 0]
110
111             #Cada resultado (posibles alertas) se inserta en el mapa
112             for migration in migrations:
113                 origin_coordinates = geohash.decode(data['geohash'])
114                 origin_coordinates_f[0] = float(origin_coordinates[0])
115                 origin_coordinates_f[1] = float(origin_coordinates[1])
116
117                 destiny_coordinates = geohash.decode(migration["b.location"])
118                 destiny_coordinates_f[0] = float(destiny_coordinates[0])
119                 destiny_coordinates_f[1] = float(destiny_coordinates[1])
120                 coordinates= (origin_coordinates_f,destiny_coordinates_f)
121                 folium.PolyLine(
122                     locations=coordinates,
123                     color= mapcolors[migration['type(r)']],
```

```
124                     weight= 1,
125                     ).add_to(map)
126                 popup_message = """
127                     <b>Destiny location:</b> {}<br>
128                     <b>Destiny municipality:</b> {}<br>
129                     <b>Destiny province:</b> {}<br>
130                     <b>Sightings:</b> {} <br>
131                     """.format(migration["b.localidad"], migration["b.municipio"],
132                                 migration["b.provincia"], migration["r.valor"])
133                 folium.Circle(
134                     location= destiny_coordinates_f,
135                     radius = 25000,
136                     fill=True, # Set fill to True
137                     fill_color='orange',
138                     color = 'orange',
139                     popup=folium.Popup(popup_message, max_width=400),
140                     fill_opacity=math.log(migration["r.valor"]),
141                     ).add_to(map)
142
143     # Por último se añade la leyenda...
144     legend_html = '''
145      <div style="position: fixed;
146      bottom: 50px; left: 50px; width: 140px; height: 130px;
147      border:2px solid grey; z-index:9999; font-size:12px;
148      ">  <b>Legend</b> <br>
149        Outbreak   <i class="fa fa-circle fa-2x"
150                  style="color:#339af0"></i><br>
151        Alert   <i class="fa fa-circle fa-2x"
152                  style="color:orange"></i><br>
153        Ciconia Migration  
154      <i class="fa fa-minus fa-2x" style="color:#04B404"></i><br>
155        Anser Migration   <i class="fa fa-minus fa-2x"
156                  style="color:#FE2E64"></i><br>
157       </div>
158      '''
159
160     map.get_root().html.add_child(folium.Element(legend_html))
161     # ...Y se guarda el mapa renderizado
162     map.save('caballesTFM/templates/public/folium_alert_map.html')
```

Apéndice C

Manual de instalación y ejecución

La herramienta requiere de un servidor con una IP pública para ser alojada. No existen requerimientos de Sistema Operativo para correr la aplicación, pero en el presente anexo se va a indicar el proceso necesario para instalar y ejecutar la aplicación en un sistema Ubuntu.

C.1. Requisitos de software previos

Antes de arrancar la aplicación es necesario tener instalado el siguiente software:

- **Python3**:

```
$ sudo apt-get install python3
```

- Instalador de paquetes **Pip3**, este proceso puede tardar unos minutos:

```
$ sudo apt-get install python3-pip
```

- **MongoDB**, para instalar esta base de datos hay que ejecutar los siguientes cuatro comandos:

```
$   wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc
| sudo apt-key add -
$ echo "deb [ arch=amd64 ]
https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse"
| sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
$ sudo apt-get update
$ sudo apt-get install -y mongodb-org
```

- **Neo4j**, para el cual además se necesita una versión de Java 8 instalada.

```
$ sudo add-apt-repository -y ppa:openjdk-r/ppa
$ sudo apt-get update

$ wget -O - https://debian.neo4j.org/neotechnology.gpg.key |
sudo apt-key add -
$ echo 'deb https://debian.neo4j.org/repo stable/' |
sudo tee -a /etc/apt/sources.list.d/neo4j.list
$ sudo apt-get update
$ sudo apt-get install neo4j=1:3.5.9
```

C.2. Instalación de librerías

Para facilitar el despligue se proporciona junto a la aplicación un fichero **requirements.txt**. Basta con ejecutar el siguiente comando para instalar Flask, así como todas las librerías Python utilizadas:

```
$ sudo pip3 install -r requirements.txt
```

Además para evitar un problema de compatibilidades con una de las librerías encargadas del web scraping se debe ejecutar:

```
$ sudo pip3 install --upgrade requests
```

Existe un bug conocido en la librería Geohash, para solucionarlo basta con cambiar el nombre del paquete de Geohash a geohash después cambiar el fichero __init__.py e importar desde .geohash (con el punto delante).

La lista de los posibles directorios donde pueden estar instalados los paquetes en Python puede verse ejecutando en consola:

```
$ python3 -m site
```

C.3. Arranque de los servicios

Por último se han de arrancar y configurar los servicios necesarios y configurar el script que actualiza la base de datos todas las noches.

Para arrancar MongoDB y neo4j se han de ejecutar los comandos:

```
$ sudo service mongod start
$ sudo service neo4j start
```

Una vez arrancados los servicios de las bases de datos es el momento de poblarlas. Para ello se ejecutarán los diferentes programas proporcionados dentro del directorio scripts:

```
# Carga del histórico de brotes a MongoDB , puede tardar varias horas
$ python3 scraping.py
```

```
# Carga del histórico de migraciones desde CSV a MongoDB
$ python3 migrations.py
# Generación del código de geohash en las anteriores colecciones
$ python3 geohashing.py
# Generación código para Neo4j
$ python3 neo4j.py
```

El resultado de la ejecución del último comando es un bloque de texto que ha de copiarse en la consola de de Neo4j para generar el grafo de migraciones. Puede accederse a dicha consola desde la terminal web o desde el navegador a través del puerto 7687.

El último servicio a configurar es el demonio *crontab* para que todas las noches ejecute el script de scrapeo de los brotes. Para ello se ha de ejecutar el comando:

```
$ crontab -e
```

Este comando abrirá un editor de texto en el que se debe añadir la línea:

```
0 1 * * * python3 /home/caballesTFM/scripts/script_auto.py
```

Si se han seguido los anteriores pasos ya está todo listo para lanzar el servicio de la herramienta, para lo cual se ha de ejecutar:

```
sudo flask run --host=0.0.0.0 --port=80 &
```